NASA-CR-192298

MCAT Institute
Progress Report
93-03

DEVELOPMENT OF COMPUTATIONAL METHODS FOR HEAVY LIFT LAUNCH VEHICLES

Seokkwan Yoon and James S. Ryan

February 1993

NCC2-505

MCAT Institute
3933 Blue Gum Drive
San Jose, CA 95127

Development of Computational Methods for National Launch System

by
Seokkwan Yoon

The research effort has been focused on the development of an advanced flow solver for complex viscous turbulent flows with shock waves.

The three-dimensional Euler and full/thin-layer Reynolds-averaged Navier-Stokes equations for compressible flows are solved on structured hexahedral grids. The Baldwin-Lomax algebraic turbulence model is used for closure. The space discretization is based on a cell-centered finite-volume method augmented by a variety of numerical dissipation models with optional total variation diminishing limiters. The governing equations are integrated in time by an implicit method based on lower-upper factorization and symmetric Gauss-Seidel relaxation. The algorithm is vectorized on diagonal planes of sweep using two-dimensional indices in three dimensions.

A new computer program named *CENS3D* has been developed for viscous turbulent flows with discontinuities. Details of the code are described in Appendix A and Appendix B.

With the developments of the numerical algorithm and dissipation model, the simulation of three-dimensional viscous compressible flows has become more efficient and accurate. The results of the research are expected to yield a direct impact on the design process of future liquid fueled launch systems.

APPENDIX A

Appendix A

"Solution of Three-Dimensional Navier-Stokes Equations Using an Implicit Gauss-Seidel Scheme," The 13th International Conference on Numerical Methods in Fluid Dynamics, July 1992.

Paper for the 13th International Conference On
Numerical Methods in Fluid Dynamics
Rome, Italy, July 6-10, 1992

SOLUTION OF THREE-DIMENSIONAL NAVIER-STOKES EQUATIONS USING AN IMPLICIT GAUSS-SEIDEL SCHEME

S. Yoon

MCAT Institute
MS 258-1, Moffett Field, California 94035, U.S.A.

I. Introduction

Although unstructured grid methods have been used successfully in solving the Euler equations for complex geometries, structured zonal grid solvers still remain the most useful for the Navier-Stokes equations because of their natural advantages in dealing with the highly clustered meshes in the viscous boundary layers. Zonal structured grid methods not only handle reasonably complex geometries using multiple blocks, but also offer a hybrid grid scheme to alleviate difficulties which unstructured grid methods have encountered. Recent developments in structured grid solvers have been focused on the efficiency as well as the accuracy since existing three-dimensional Navier-Stokes codes are not efficient enough to be used routinely for aerodynamic design.

The author[1] has introduced an implicit algorithm based on a lower-upper factorization and symmetric Gauss-Seidel relaxation. The scheme has been used successfully in computing chemically reacting flows due in part to the algorithm's property which reduces the size of the left hand side matrix for nonequilibrium flows with finite rate chemistry.[2,3] More recently, a study[4] suggests that the three-dimensional extension of the method is one of the most efficient ways to solve the Navier-Stokes equations. Consequently, a new three-dimensional Navier-Stokes code named CENS3D was produced. CENS3D requires less computational work per iteration than most existing codes on a Cray YMP supercomputer and in addition converges reasonably fast. The performance of the code is demonstrated for a viscous transonic flow past an ONERA M6 wing.

II. Numerical Methods

Let t be time; $\widehat{Q}$ the vector of conserved variables; $\widehat{E}$, $\widehat{F}$, and $\widehat{G}$ the convective flux vectors; and $\widehat{E}_v$, $\widehat{F}_v$, and $\widehat{G}_v$ the flux vectors for the viscous terms. Then the three-dimensional Navier-Stokes equations in generalized curvilinear coordinates (ξ, η, ζ) can be written as

$$\partial_t \widehat{Q} + \partial_\xi(\widehat{E} - \widehat{E}_v) + \partial_\eta(\widehat{F} - \widehat{F}_v) + \partial_\zeta(\widehat{G} - \widehat{G}_v) = 0 \tag{1}$$

where the flux vectors are found in Ref. 4.

An unfactored implicit scheme can be obtained from a nonlinear implicit scheme by linearizing the flux vectors about the previous time step and dropping terms of the second and higher order.

$$[I + \alpha\Delta t(D_\xi \hat{A} + D_\eta \hat{B} + D_\zeta \hat{C})]\delta\hat{Q} = -\Delta t\hat{R} \tag{2}$$

where $\hat{R}$ is the residual

$$\hat{R} = D_\xi(\hat{E} - \hat{E}_v) + D_\eta(\hat{F} - \hat{F}_v) + D_\zeta(\hat{G} - \hat{G}_v) \tag{3}$$

and I is the identity matrix. $\delta\hat{Q}$ is the correction $\hat{Q}^{n+1}$ - $\hat{Q}^n$, where n denotes the time level. D_ξ, D_η, and D_ζ are difference operators that approximate ∂_ξ, ∂_η, and ∂_ζ. $\hat{A}$, $\hat{B}$, and $\hat{C}$ are the Jacobian matrices of the convective flux vectors.

An efficient implicit scheme can be derived by combining the advantages of LU factorization and Gauss-Seidel relaxation.

$$LD^{-1}U\delta\hat{Q} = -\Delta t\hat{R} \tag{4}$$

Here,

$$\begin{aligned} L &= I + \alpha\Delta t(D_\xi^- \hat{A}^+ + D_\eta^- \hat{B}^+ + D_\zeta^- \hat{C}^+ - \hat{A}^- - \hat{B}^- - \hat{C}^-) \\ D &= I + \alpha\Delta t(\hat{A}^+ - \hat{A}^- + \hat{B}^+ - \hat{B}^- + \hat{C}^+ - \hat{C}^-) \\ U &= I + \alpha\Delta t(D_\xi^+ \hat{A}^- + D_\eta^+ \hat{B}^- + D_\zeta^+ \hat{C}^- + \hat{A}^+ + \hat{B}^+ + \hat{C}^+) \end{aligned} \tag{5}$$

where D_ξ^-, D_η^-, and D_ζ^- are backward difference operators, while D_ξ^+, D_η^+, and D_ζ^+ are forward difference operators.

In the framework of the LU-SGS algorithm, a variety of schemes can be developed by different choices of numerical dissipation models and Jacobian matrices of the flux vectors. Jacobian matrices leading to diagonal dominance are constructed so that " + " matrices have nonnegative eigenvalues while " - " matrices have nonpositive eigenvalues. For example,

$$\begin{aligned} \hat{A}^\pm &= \hat{T}_\xi \Lambda_\xi^\pm \hat{T}_\xi^{-1} \\ \hat{B}^\pm &= \hat{T}_\eta \Lambda_\eta^\pm \hat{T}_\eta^{-1} \\ \hat{C}^\pm &= \hat{T}_\zeta \Lambda_\zeta^\pm \hat{T}_\zeta^{-1} \end{aligned} \tag{6}$$

where $\hat{T}_\xi$ and $\hat{T}_\xi^{-1}$ are similarity transformation matrices of the eigenvectors of $\hat{A}$. Another possibility is to construct Jacobian matrices of the flux vectors approximately to yield diagonal dominance.

$$\begin{aligned} \hat{A}^\pm &= \frac{1}{2}[\hat{A} \pm \tilde{\rho}(\hat{A})I] \\ \hat{B}^\pm &= \frac{1}{2}[\hat{B} \pm \tilde{\rho}(\hat{B})I] \\ \hat{C}^\pm &= \frac{1}{2}[\hat{C} \pm \tilde{\rho}(\hat{C})I] \end{aligned} \tag{7}$$

where

$$\tilde{\rho}(\hat{A}) = \kappa \; max[|\, \lambda(\hat{A}) \,|] \tag{8}$$

for example. Here $\lambda(\hat{A})$ represent eigenvalues of the Jacobian matrix $\hat{A}$ and κ is a constant that is greater than or equal to 1. Stability and convergence are controlled by adjusting κ either manually or automatically as the flowfield develops.

It is interesting to note that the need for block inversions along the diagonals can be eliminated if we use the approximate Jacobian matrices of Eq. (7). Setting $\alpha = 1$ and $\Delta t = \infty$ yields a Newton-like iteration. Although a quadratic convergence of the Newton method cannot be achieved because of the approximate factorization, a linear convergence can be demonstrated. The use of Newton-like iteration offers a practical advantage in that one does not have to find an optimal Courant number or time step to reduce the overall computer time.

The cell-centered finite-volume method[4] is augmented by a numerical dissipation model with a minmod flux limiter. The coefficients of the dissipative terms are the directionally scaled spectral radii of Jacobian matrices.

III. Results

In order to demonstrate the performance of the CENS3D code, transonic flow calculations have been carried out for an ONERA M6 wing. A 289 x 50 x 44 C-H mesh (635,800 points) is used as a fine grid. The distance of the first grid point from the wing surface is 1.0 x 10^{-5} times the chord length at the root section. The freestream conditions are at a Mach number of 0.8395, Reynolds number of 1.5 x 10^7, and a 3.06° angle of attack. This is an unseparated flow case. The algebraic turbulence model by Baldwin and Lomax is employed for mathematical closure of the Reynolds-averaged Navier-Stokes equations. The root-mean-squared residuals drop 3 orders of magnitude in about 380 iterations or 38 minutes of CPU time on the fine grid. In the present implementation, the implicit left hand side viscous terms are not included which decreases the computational work per iteration. To investigate the effect of this left hand side compromise on the convergence rate, a grid-convergence study has been performed using a 171 x 25 x 44 (188,100 points) coarse grid. Although the number of radial grid points to resolve the viscous boundary layer is doubled in the fine grid case, the fine grid convergence is slowed by only twenty percent. Fig. 1 and Fig. 2 show a good agreement between experimental data[5] and the pressure coefficients at 44% and 65% semi-span stations computed on the fine grid. This comparison validates the present code CENS3D.

The CENS3D code requires only 9 μsec per grid-point per iteration for the thin-layer Navier-Stokes equations with an algebraic turbulence model on a single Cray YMP processor at the sustained rate of 175 Mflops. It is interesting to note that the LU-SGS implicit scheme requires less computational work per iteration than a Runge-Kutta explicit scheme.

Conclusions

Good performance of a three-dimensional Navier-Stokes solver CENS3D based on an implicit lower-upper Gauss-Seidel scheme is demonstrated for nonseparated transonic flow past a wing. In addition to its reasonabe convergence rate, the code requires very low computational time per iteration. The three-dimensional Navier-Stokes solution of a high Reynolds number flow using 636K grid points is obtained in 38 minutes. The computational results compare well with available experimental data.

References

1. Yoon, S. and Jameson, A., "Lower-Upper Symmetric-Gauss-Seidel Method for the Euler and Navier-Stokes Equations," *AIAA Journal*, Vol. 26, Sep. 1988, pp. 1025-1026.

2. Shuen, J.S. and Yoon, S., "A Numerical Study of Chemically Reacting Flows Using a Lower-Upper Symmetric Successive Overrelaxation Scheme," *AIAA Journal*, Vol. 27, Dec. 1989, pp. 1752-1760.

3. Park, C. and Yoon, S., "A Fully-Coupled Implicit Method for Thermo-Chemical Nonequilibrium Air at Sub-Orbital Flight Speeds," *Journal of Spacecraft and Rockets*, Vol. 28, No. 1, Jan.-Feb. 1991, pp. 31-39.

4. Yoon, S. and Kwak, D., "An Implicit Three-Dimensional Navier-Stokes Solver For Compressible Flows," AIAA Paper 91-1555, June 1991.

5. Schmitt, V. and Charpin, F., " Pressure Distributions on the ONERA M6 Wing at Transonic Mach Numbers," AGARD AR-138-B1, 1979.

APPENDIX B

Appendix B

"An Implicit Navier-Stokes Solver For Three-Dimensional Compressible Flows," *AIAA Journal* Dec. 1992 (to appear).

AN IMPLICIT NAVIER-STOKES SOLVER FOR THREE-DIMENSIONAL COMPRESSIBLE FLOWS

Seokkwan Yoon *

MCAT Institute

Moffett Field, California

and

Dochan Kwak †

NASA Ames Research Center

Moffett Field, California

Abstract

A three-dimensional numerical method based on the lower-upper symmetric-Gauss-Seidel implicit scheme in conjunction with the flux-limited dissipation model is developed for solving the compressible Navier-Stokes equations. A new computer code which is based on this method requires only 9 μsec per grid-point per iteration on a single processor of a Cray YMP computer and executes at the sustained rate of 175 MFLOPS. A reduction of three orders of magnitude in the residual for a high Reynolds number flow using 636K grid points is obtained in 38 minutes. The computational results compare well with available experimental data.

I. Introduction

* Senior Member AIAA

† Associate Fellow AIAA

Since the computational requirements for direct simulation of turbulent flows about complex three-dimensional geometries are still beyond the reach of the most powerful supercomputers, most numerical algorithms developed so far focus on the solution of the Reynolds-averaged Navier-Stokes equations, which can be obtained by ensemble-averaging of rapidly fluctuating components. The governing equations of fluid flows can be integrated by either explicit or implicit methods. Although explicit schemes have been successful in solving the Euler equations for inviscid flows, the efficiency of explicit schemes in solving the Navier-Stokes equations is limited by the Courant-Friedrichs-Lewy condition, which is especially restrictive when the computational grid is highly clustered to resolve the viscous boundary layer. When the time step limit imposed by an explicit stability bound is significantly less than the accuracy requirement, implicit schemes are often preferred. However, the trade-off between a decreased number of iterations and an increased operation count per iteration for the implicit methods must be considered. The fastest convergence rate may be attained by an unfactored implicit scheme which directly inverts a large block banded matrix using Gaussian elimination. Such a scheme is impractical in three-dimensions because of the rapid increase of the number of operations as the number of mesh points increases and because of the large memory requirement.

Yoon and Jameson[1-3] introduced an implicit algorithm based on a lower-upper factorization and Gauss-Seidel relaxation for the Euler and Navier-Stokes equations. Since then, the lower-upper symmetric-Gauss-Seidel (LU-SGS) scheme has been successfully implemented by many researchers. Shuen and Yoon[4] applied the method to supersonic combustion ramjet problems for the National Aero-Space Plane to take advantage of the algorithm's property that reduces the size of matrix for reacting flows with finite rate chemistry. The resulting computer program RPLUS was named after the original perfect gas code PLUS (Program using LU Schemes).[1] A variation of the PLUS code named IPLUS was applied to internal flows through turbomachinery cascades in conjunction with an interactive grid generation technique by Choo, Soh, and Yoon.[5] Another variant named HPLUS demonstrated the robustness of an LU scheme at high Mach numbers.[6] Rieger and Jameson[7] developed a three-dimensional code based on an early version of the PLUS code and applied it to Hermes, the European space shuttle. Yu, Tsai, and Shuen[8] extended the RPLUS code to three-dimensions. Coirier[9] developed a finite difference version of the RPLUS code for corner and gap-seal calculations. However, the accuracy and efficiency of the above codes have been limited by the

artificial viscosity model.[10]

Yoon and Kwak[11,12] proposed that a variety of schemes could be constructed in the framework of the LU-SGS algorithm by different choices of Jacobian matrices of flux vectors and numerical dissipation models. The computer code CENS2D (Compressible Euler and Navier-Stokes) was written to study the effects of different dissipation models. It was observed that the blended first and third order model was the least accurate while the flux-difference split upwind-biased model was not only the most expensive but the least robust when the grid lines were not aligned with strong bow shock waves. It was concluded in the study that the flux-limited dissipation model was a practical alternative to upwind schemes because of its robustness, efficiency and accuracy for high speed external flows. Recently, promising results were reported using upwind-biased and total variation diminishing schemes with the LU-SGS implicit scheme. They include Obayashi[13] for underexpanded plumes, Chen, McCrosky, and Obayashi[14] for forward-flight rotor flow, Loh and Golafshani[15] for flows in hybrid rocket motors, Yungster[16] for shock wave and boundary layer interactions, and Imlay and Eberhardt[17] for flows past the Aeroassist Flight Experiment vehicle. In the meantime, the CENS2D code has been extended by Park and Yoon[18-20] to compute thermo-chemical nonequilibrium in hypersonic external flows using a multiple temperature model.

While conventional implicit methods often achieve fast convergence rates, they suffer from greater computer time per iteration than explicit methods. The LU-SGS implicit scheme offers a potential for very low computer time per iteration as well as fast convergence. High efficiency can be achieved by accomplishing the complete vectorizability of the algorithm on oblique planes of sweep in three-dimensions.[21] It has been demonstrated that the LU-SGS scheme requires less computational work per iteration than most existing schemes on a Cray YMP supercomputer in the case of three-dimensional viscous incompressible flows. One of the objectives of the present work is to provide standard performance figures which the LU-SGS scheme can achieve for three-dimensional compressible flows in conjunction with the flux-limited dissipation model by developing a new testbed code named CENS3D.

II. The Navier-Stokes Equations

Let t be time; ρ, p, and T the density, pressure, and temperature; u, v, and w the velocity components in Cartesian coordinates (x, y, z); $\widehat{Q}$ the vector of conserved variables; $\widehat{E}$, $\widehat{F}$, and $\widehat{G}$ the convective flux vectors; and $\widehat{E}_v$, $\widehat{F}_v$, and $\widehat{G}_v$ the flux vectors for the viscous terms. Then the three-dimensional Navier-Stokes equations in generalized curvilinear coordinates $(\xi,\ \eta,\ \zeta)$ can be written as

$$\partial_t \widehat{Q} + \partial_\xi(\widehat{E} - \widehat{E}_v) + \partial_\eta(\widehat{F} - \widehat{F}_v) + \partial_\zeta(\widehat{G} - \widehat{G}_v) = 0 \tag{1}$$

The flux vectors for compressible and incompressible flows are different. The flux vectors for compressible flow are

$$\widehat{Q} = h\begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{bmatrix}, \qquad \widehat{E} = h\begin{bmatrix} \rho U \\ \rho U u + \xi_x p \\ \rho U v + \xi_y p \\ \rho U w + \xi_z p \\ U(e+p) \end{bmatrix}$$

$$\widehat{F} = h\begin{bmatrix} \rho V \\ \rho V u + \eta_x p \\ \rho V v + \eta_y p \\ \rho V w + \eta_z p \\ V(e+p) \end{bmatrix}, \qquad \widehat{G} = h\begin{bmatrix} \rho W \\ \rho W u + \zeta_x p \\ \rho W v + \zeta_y p \\ \rho W w + \zeta_z p \\ W(e+p) \end{bmatrix} \tag{2}$$

where e is the total energy. The contravariant velocity components U, V, and W are defined as

$$\begin{aligned} U &= \xi_x u + \xi_y v + \xi_z w \\ V &= \eta_x u + \eta_y v + \eta_z w \\ W &= \zeta_x u + \zeta_y v + \zeta_z w \end{aligned} \tag{3}$$

The equation of state is needed to complete the set of equations for compressible flow.

$$p = (\gamma - 1)[e - \frac{1}{2}\rho(u^2 + v^2 + w^2)] \tag{4}$$

where γ is the ratio of specific heats. Here, h is the determinant of the inverse of transformation Jacobian matrix.

$$h = \begin{vmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{vmatrix} \tag{5}$$

The flux vectors for incompressible flow can be written in a similar way if the pseudocompressibility formulation[21] is used. In a finite volume formulation, h is identical to the mesh cell volume. The viscous flux vectors are

$$\begin{aligned} \widehat{E}_v &= h\left[\xi_x E_v + \xi_y F_v + \xi_z G_v\right] \\ \widehat{F}_v &= h\left[\eta_x E_v + \eta_y F_v + \eta_z G_v\right] \\ \widehat{G}_v &= h\left[\zeta_x E_v + \zeta_y F_v + \zeta_z G_v\right] \end{aligned} \tag{6}$$

Their Cartesian components are

$$\begin{aligned} E_v &= \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + k\partial_x T \end{bmatrix} \\ F_v &= \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + k\partial_y T \end{bmatrix} \\ G_v &= \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + k\partial_z T \end{bmatrix} \end{aligned} \tag{7}$$

where

$$\begin{aligned} \tau_{xx} &= 2\mu\partial_x u - \frac{2}{3}\mu(\partial_x u + \partial_y v + \partial_z w) \\ \tau_{yy} &= 2\mu\partial_y v - \frac{2}{3}\mu(\partial_x u + \partial_y v + \partial_z w) \\ \tau_{zz} &= 2\mu\partial_z w - \frac{2}{3}\mu(\partial_x u + \partial_y v + \partial_z w) \\ \tau_{xy} &= \tau_{yx} = \mu(\partial_y u + \partial_x v) \\ \tau_{xz} &= \tau_{zx} = \mu(\partial_z u + \partial_x w) \\ \tau_{yz} &= \tau_{zy} = \mu(\partial_z v + \partial_y w) \end{aligned} \tag{8}$$

Here the coefficient of viscosity μ and the coefficient of thermal conductivity k are decomposed into laminar and turbulent contributions.

$$\mu = \mu_l + \mu_t \tag{9}$$

$$k = \frac{\gamma}{\gamma - 1}\left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t}\right) \tag{10}$$

where Pr_l and Pr_t denote laminar and turbulent Prandtl numbers.

While the Euler equations can be obtained by neglecting the viscous terms, the thin-layer Navier-Stokes equations can be obtained by retaining the viscous flux vector in the direction normal to body surfaces.

III. Implicit Methods

The governing equations are integrated in time for both steady and unsteady flow calculations. For a steady-state solution, the use of a large time step leads to fast convergence. For a time-accurate solution, it is desirable that the time step is determined by the physics rather than the numerics. An unfactored implicit scheme can be obtained from a nonlinear implicit scheme by linearizing the flux vectors about the previous time step and dropping terms of the second and higher order.

$$[I + \alpha\Delta t(D_\xi \hat{A} + D_\eta \hat{B} + D_\zeta \hat{C})]\delta\hat{Q} = -\Delta t \hat{R} \tag{11}$$

where $\hat{R}$ is the residual

$$\hat{R} = D_\xi(\hat{E} - \hat{E}_v) + D_\eta(\hat{F} - \hat{F}_v) + D_\zeta(\hat{G} - \hat{G}_v) \tag{12}$$

and I is the identity matrix. $\delta\hat{Q}$ is the correction $\hat{Q}^{n+1}$ - $\hat{Q}^n$, where n denotes the time level. D_ξ, D_η, and D_ζ are difference operators that approximate ∂_ξ, ∂_η, and ∂_ζ. $\hat{A}$, $\hat{B}$, and $\hat{C}$ are the Jacobian matrices of the convective flux vectors.

$$\hat{A} = \frac{\partial \hat{E}}{\partial \hat{Q}}, \quad \hat{B} = \frac{\partial \hat{F}}{\partial \hat{Q}}, \quad \hat{C} = \frac{\partial \hat{G}}{\partial \hat{Q}} \tag{13}$$

For compressible flow,

$$\hat{A} = \begin{bmatrix} 0 & \xi_x \\ \xi_x \tilde{q} - Uu & U - \xi_x u(\gamma - 2) \\ \xi_y \tilde{q} - Uv & \xi_x v - \xi_y u(\gamma - 1) \\ \xi_z \tilde{q} - Uw & \xi_x w - \xi_z u(\gamma - 1) \\ U(\tilde{q} - \tilde{h}) & \xi_x \tilde{h} - Uu(\gamma - 1) \end{bmatrix}$$

(continued)

$$\begin{bmatrix} \xi_y & \xi_z & 0 \\ \xi_y u - \xi_x v(\gamma - 1) & \xi_z u - \xi_x w(\gamma - 1) & \xi_x(\gamma - 1) \\ U - \xi_y v(\gamma - 2) & \xi_z v - \xi_y w(\gamma - 1) & \xi_y(\gamma - 1) \\ \xi_y w - \xi_z v(\gamma - 1) & U - \xi_z w(\gamma - 2) & \xi_z(\gamma - 1) \\ \xi_y \tilde{h} - Uv(\gamma - 1) & \xi_z \tilde{h} - Uw(\gamma - 1) & U\gamma \end{bmatrix}$$

where

$$\tilde{q} = \frac{\gamma - 1}{2}(u^2 + v^2 + w^2) \tag{14}$$

$$\tilde{h} = \frac{e + p}{\rho} \tag{15}$$

Matrices $\hat{B}$ and $\hat{C}$ are similarly derived. Although the direct inversion method seems to be competitive with approximate factorization methods in the overall computing time in two-dimensions,[22] direct inversion of a large block banded matrix of the unfactored scheme Eq. (11) appears to be impractical in three-dimensions as stated before.

To alleviate this difficulty, many investigators have focused on indirect methods. The popular Alternating Direction Implicit (ADI) scheme by Beam and Warming[23] or Briley and McDonald[24] replaces the implicit operator of the unfactored scheme by a product of three one-dimensional operators.

$$(I + \alpha \Delta t D_\xi \hat{A})(I + \alpha \Delta t D_\eta \hat{B})(I + \alpha \Delta t D_\zeta \hat{C}) \delta \hat{Q} = -\Delta t \hat{R} \tag{16}$$

The ADI scheme which is unconditionally stable in two-dimensions becomes unstable in three-dimensions, although numerical dissipation conditionally stabilizes the method. Due to three factors, the ADI scheme also introduces the error terms of $(\Delta t)^3$. The large factorization error associated with this scheme further reduces the rate of convergence. In spite of these drawbacks, the ADI scheme has been successful due to the reduction of cost by the diagonalization of Jacobian matrices by Pulliam and Chaussee.[25] Obayashi and Kuwahara[26] developed a scheme by replacing each factor with bidiagonal LU factors.

$$(I + \alpha\Delta t D_\xi^- \hat{A}^+)(I + \alpha\Delta t D_\xi^+ \hat{A}^-)(I + \alpha\Delta t D_\eta^- \hat{B}^+)(I + \alpha\Delta t D_\eta^+ \hat{B}^-)$$

$$(I + \alpha\Delta t D_\zeta^- \hat{C}^+)(I + \alpha\Delta t D_\zeta^+ \hat{C}^-)\delta\hat{Q} = -\Delta t \hat{R} \tag{17}$$

Stability and convergence characteristics of the LU-ADI scheme appear to be similar to the ADI scheme.

The factorization errors of two-factor schemes, which are of order $(\Delta t)^2$, are lower than the ADI scheme. Two-factor schemes can also be stable in three-dimensions. Steger proposed a two-factor scheme[27,28] by partially splitting the flux vectors.

$$[I + \alpha\Delta t(D_\xi^- \hat{A}^+ + D_\eta \hat{B})][I + \alpha\Delta t(D_\xi^+ \hat{A}^- + D_\zeta \hat{C})]\delta\hat{Q} =$$

$$-\Delta t(D_\xi^- \hat{E}^+ + D_\xi^+ \hat{E}^- + D_\eta \hat{F} + D_\zeta \hat{G}) \tag{18}$$

The scheme was incorporated in F3D code[28,29] and CNS code.[30] The partially flux-split scheme is more expensive than the diagonalized ADI scheme because of block tridiagonal inversions.

An alternative two-factor scheme is based on an lower-upper(LU) factorization proposed by Steger and Warming[27] and Jameson and Turkel[31].

$$LU\delta\hat{Q} = -\Delta t \hat{R} \tag{19}$$

where

$$L = I + \alpha\Delta t(D_\xi^- \hat{A}^+ + D_\eta^- \hat{B}^+ + D_\zeta^- \hat{C}^+)$$
$$U = I + \alpha\Delta t(D_\xi^+ \hat{A}^- + D_\eta^+ \hat{B}^- + D_\zeta^+ \hat{C}^-) \tag{20}$$

where D_ξ^-, D_η^-, and D_ζ^- are backward difference operators, while D_ξ^+, D_η^+, and D_ζ^+ are forward difference operators. Despite its early introduction in the late '70s, the LU scheme had not been used until it was independently implemented by Buning and Steger[32], Whitfield[33], Buratynski and Caughey[34], and Jameson and Yoon.[2,3] The cost of the LU scheme is more expensive than the diagonalized ADI scheme because of block diagonal inversions.

MacCormack[35] introduced an implicit line relaxation method based on back-and-forth symmetric sweeps in conjunction with upwind flux splittings. Although the line Gauss-Seidel relaxation method allowed significant increase of work per iteration compared to approximate factorization schemes due to multiple block tridiagonal inversions and sequential operations, it achieved very fast convergence rates. In fact, all the implicit schemes mentioned above require much larger computational work per iteration than explicit schemes.

Yoon and Jameson[1] derived a new implicit algorithm by combining the advantages of LU factorization and SGS relaxation. The LU-SGS scheme has quite different L and U factors from those of the LU scheme. Unlike the line SGS relaxation scheme, no additional relaxation or factorization is required on planes of sweep. The LU-SGS scheme can be written as

$$LD^{-1}U\delta\hat{Q} = -\Delta t\hat{R} \tag{21}$$

where

$$
\begin{aligned}
L &= I + \alpha\Delta t(D_\xi^- \hat{A}^+ + D_\eta^- \hat{B}^+ + D_\zeta^- \hat{C}^+ - \hat{A}^- - \hat{B}^- - \hat{C}^-) \\
D &= I + \alpha\Delta t(\hat{A}^+ - \hat{A}^- + \hat{B}^+ - \hat{B}^- + \hat{C}^+ - \hat{C}^-) \\
U &= I + \alpha\Delta t(D_\xi^+ \hat{A}^- + D_\eta^+ \hat{B}^- + D_\zeta^+ \hat{C}^- + \hat{A}^+ + \hat{B}^+ + \hat{C}^+)
\end{aligned}
\tag{22}
$$

In the framework of the LU-SGS algorithm, a variety of schemes can be developed by different choices of numerical dissipation models and Jacobian matrices of the flux vectors.[11] It is desirable that the matrix should be diagonally dominant to assure convergence to a steady state. Jacobian matrices leading to diagonal dominance are constructed so that " + " matrices have nonnegative eigenvalues while " - " matrices have nonpositive eigenvalues. For example,

$$
\begin{aligned}
\hat{A}^\pm &= \hat{T}_\xi \Lambda_\xi^\pm \hat{T}_\xi^{-1} \\
\hat{B}^\pm &= \hat{T}_\eta \Lambda_\eta^\pm \hat{T}_\eta^{-1} \\
\hat{C}^\pm &= \hat{T}_\zeta \Lambda_\zeta^\pm \hat{T}_\zeta^{-1}
\end{aligned}
\tag{23}
$$

where $\hat{T}_\xi$ and $\hat{T}_\xi^{-1}$ are similarity transformation matrices of the eigenvectors of $\hat{A}$. Another possibility is to construct Jacobian matrices of the flux vectors approximately to yield diagonal dominance.

$$
\begin{aligned}
\hat{A}^\pm &= \frac{1}{2}[\hat{A} \pm \tilde{\rho}(\hat{A})I] \\
\hat{B}^\pm &= \frac{1}{2}[\hat{B} \pm \tilde{\rho}(\hat{B})I] \\
\hat{C}^\pm &= \frac{1}{2}[\hat{C} \pm \tilde{\rho}(\hat{C})I]
\end{aligned}
\tag{24}
$$

where

$$
\tilde{\rho}(\hat{A}) = \kappa \; max[|\, \lambda(\hat{A})\, |] \tag{25}
$$

for example. Here $\lambda(\hat{A})$ represent eigenvalues of the Jacobian matrix $\hat{A}$ and κ is a constant that is greater than or equal to 1. Stability and convergence can be controlled by adjusting κ either manually or automatically as the flowfield develops. The diagonal matrix of eigenvalues is

$$
\hat{\Lambda}(\hat{A}) = \begin{bmatrix} U & 0 & 0 & 0 & 0 \\ 0 & U & 0 & 0 & 0 \\ 0 & 0 & U & 0 & 0 \\ 0 & 0 & 0 & U + C_\xi & 0 \\ 0 & 0 & 0 & 0 & U - C_\xi \end{bmatrix} \tag{26}
$$

and

$$C_\xi = c\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2} \tag{27}$$

where c is the speed of sound

$$c = \sqrt{\frac{\gamma p}{\rho}} \tag{28}$$

In the early days of development of codes such as the PLUS series, the Eq. (21) was inverted in three steps as following.

$$\begin{aligned} \delta\hat{Q}^* &= -\Delta t D\hat{R} \\ \delta\hat{Q}^{**} &= L^{-1}\hat{Q}^* \\ \delta\hat{Q} &= U^{-1}\hat{Q}^{**} \end{aligned} \tag{29}$$

This is not a mathematically correct procedure, although no difference in the solution or convergence has been observed when D is a scalar diagonal matrix. The correct order used in INS3D-LU and CENS3D codes is

$$\begin{aligned} \delta\hat{Q}^* &= -\Delta t L^{-1}\hat{R} \\ \delta\hat{Q}^{**} &= D\hat{Q}^* \\ \delta\hat{Q} &= U^{-1}\hat{Q}^{**} \end{aligned} \tag{30}$$

It is interesting to note that the need for block inversions along the diagonals can be eliminated if we use the approximate Jacobian matrices of Eq. (24). Setting $\alpha = 1$ and $\Delta t = \infty$ yields a Newton-like iteration. Although a quadratic convergence of the Newton method cannot be achieved because of the approximate factorization, a linear convergence can be demonstrated. That is why the term *Newton-like* instead of *Newton* is used to distinguish the differences. The use of Newton-like iteration offers a practical advantage that one does not have to find an optimal Courant number or time step to reduce the overall computer time. If two-point one-sided differences are used, Eq. (22) reduces to

$$\begin{aligned} L &= \tilde{\rho} I - \hat{A}^+_{i-1,j,k} - \hat{B}^+_{i,j-1,k} - \hat{C}^+_{i,j,k-1} \\ D &= \tilde{\rho} I \\ U &= \tilde{\rho} I + \hat{A}^-_{i+1,j,k} + \hat{B}^-_{i,j+1,k} + \hat{C}^-_{i,j,k+1} \end{aligned} \tag{31}$$

where

$$\tilde{\rho} = \tilde{\rho}(\hat{A}) + \tilde{\rho}(\hat{B}) + \tilde{\rho}(\hat{C}) \tag{32}$$

In the inversion process, $\hat{A}^{+}_{i-1,j,k}$ is multiplied by $\delta\hat{Q}^{*}_{i-1,j,k}$, for example. The algorithm permits scalar diagonal inversions since

$$Diagonal(L \text{ or } U) = \begin{bmatrix} \tilde{\rho} & 0 & 0 & 0 & 0 \\ 0 & \tilde{\rho} & 0 & 0 & 0 \\ 0 & 0 & \tilde{\rho} & 0 & 0 \\ 0 & 0 & 0 & \tilde{\rho} & 0 \\ 0 & 0 & 0 & 0 & \tilde{\rho} \end{bmatrix} \tag{33}$$

The use of the true Jacobian matrices of Eq. (23), which might lead to a faster convergence rate, requires block diagonal inversions and hence approximately doubles the computational work per iteration. Another interesting feature of the present algorithm is that the scheme is completely vectorizable on $i+j+k = constant$ oblique planes of sweep, which is illustrated in Fig. 1. This is achieved by reordering the three-dimensional arrays into two-dimensional arrays, that is,

$$\hat{Q}(ipoint, iplane) = \hat{Q}(i, j, k) \tag{34}$$

where *iplane* is the serial number of the oblique plane to be swept, and *ipoint* is the address on that plane. The present algorithm may also be amenable to parallel processing.

IV. Numerical Dissipation

A semidiscrete finite volume method is used to ensure the final converged solution be independent of the time step and to avoid metric singularity problems. The finite volume method is based on the local flux balance of each mesh cell. For example,

$$\partial_\xi \hat{E} + \partial_\eta \hat{F} + \partial_\zeta \hat{G} =$$

$$\hat{E}_{i+\frac{1}{2},j,k} - \hat{E}_{i-\frac{1}{2},j,k} + \hat{F}_{i,j+\frac{1}{2},k} - \hat{F}_{i,j-\frac{1}{2},k} + \hat{G}_{i,j,k+\frac{1}{2}} - \hat{G}_{i,j,k-\frac{1}{2}} \tag{35}$$

A central difference scheme achieves the second order accuracy in the most efficient way when the flow field is free of discontinuous solutions. However, numerical dissipation models are added to nondissipative central difference schemes in order to suppress the tendency for odd and even point decoupling. Dissipation models are often called *filters* since they work like low pass filters which damp out high frequency modes. The dissipative flux d is added to the convective flux in a conservative manner.

$$(\widehat{E}_{i+\frac{1}{2},j,k} - \widehat{E}_{i-\frac{1}{2},j,k} + \widehat{F}_{i,j+\frac{1}{2},k} - \widehat{F}_{i,j-\frac{1}{2},k} + \widehat{G}_{i,j,k+\frac{1}{2}} - \widehat{G}_{i,j,k-\frac{1}{2}})$$

$$-(d_{i+\frac{1}{2},j,k} - d_{i-\frac{1}{2},j,k} + d_{i,j+\frac{1}{2},k} - d_{i,j-\frac{1}{2},k} + d_{i,j,k+\frac{1}{2}} - d_{i,j,k-\frac{1}{2}}) \tag{36}$$

For simplicity, $d_{i+\frac{1}{2},j,k}$ is denoted by $d_{i+\frac{1}{2}}$ hereafter.

It has long been recognized that charateristic-based upwind-biased schemes can demonstrate crisp resolution of discontinuities. This is especially so when the flux-difference splitting scheme replaces Godunov's exact solution of the Riemann problem with an approximate solution, while distinguishing between the influence of forward and backward moving waves. High-order upwind schemes can be constructed by using multipoint extrapolation formulas to estimate the numerical flux, or by adding higher-order dissipative terms. In either case flux limiters are then added to control the signs of the coefficients of a semi-discrete approximation to the hyperbolic system of equations. The dissipative coefficient for a system of equations must be a matrix to meet the requirement of upwinding. It is sometimes necessary to add artificial dissipation in the form of entropy correction to avoid instabilities. Considering the additional cost and reduced robustness of the upwind-biased scheme when the grid lines are not aligned with strong shock waves,[11] it seems that the flux-limited dissipation model with scalar coefficients can be a practical alternative to upwind dissipation with matrix coefficients, especially when the uncertainty of the solution due to a turbulence model is relatively large.

In the flux-limited dissipation model, the dissipative flux is constructed by introducing flux limiters into the high order terms instead of adding low order terms.

$$d_{i+\frac{1}{2}} = -\alpha_{i+\frac{1}{2}}[\phi(\sigma_{i+1})e_{i+\frac{3}{2}} - 2e_{i+\frac{1}{2}} + \psi(\sigma_i)e_{i-\frac{1}{2}}] \tag{37}$$

where ϕ and ψ are flux limiting functions to limit antidiffusive fluxes

$$\phi(\sigma) = \begin{bmatrix} 0 \; if \; \sigma < 0 \\ \sigma \; if \; 0 \leq \sigma \leq 1 \\ 1 \; if \; \sigma > 1 \end{bmatrix} \tag{38}$$

and

$$\psi(\sigma) = \phi(\frac{1}{\sigma}) \tag{39}$$

Here,

$$\sigma_i = \frac{e_{i-\frac{1}{2}}}{e_{i+\frac{1}{2}}} \tag{40}$$

and

$$e_{i+\frac{1}{2}} = \widehat{Q}_{i+1} - \widehat{Q}_i \tag{41}$$

If we write $\sigma = \frac{b}{a}$, then

$$\phi(\sigma)a = minmod(a, b) \tag{42}$$

where minmod(a, b) is zero if a and b have opposite signs, and minmod(a, b) is the smaller of a and b if a and b have the same sign.

$$\alpha_{i+\frac{1}{2}} = (\kappa_0 + \kappa_1 \hat{\nu}_{i+\frac{1}{2}}) r(\widehat{A})_{i+\frac{1}{2}} \tag{43}$$

where the constant κ_0 determines a threshold, and the constant κ_1 is chosen to ensure that there is enough dissipation to suppress numerical oscillations in the neighborhood of shock waves. $r(\widehat{A})$ denotes the spectral radius of the Jacobian matrix $\widehat{A}$ and $\hat{\nu}_{i+\frac{1}{2}}$ is a sensor.

$$\hat{\nu}_{i+\frac{1}{2}} = max(\nu_{i+1}, \nu_i) \tag{44}$$

where

$$\nu_i = max(\nu_i^p, \nu_i^T) \tag{45}$$

$$\nu_i^p = \mid p_{i+1} - 2p_i + p_{i-1} \mid /(p_{i+1} + 2p_i + p_{i-1}) \tag{46}$$

$$\nu_i^T = \mid T_{i+1} - 2T_i + T_{i-1} \mid /(T_{i+1} + 2T_i + T_{i-1}) \tag{47}$$

Here p and T are the pressure and the temperature.

V. Results

The LU-SGS algorithm can be completely vectorized and its efficiency is demonstrated by the CENS3D code on a Cray YMP supercomputer at NASA Ames Research Center. The CENS3D code requires only 9 μsec per grid-point per iteration for the thin-layer option of the Navier-Stokes equations with an algebraic turbulence model on a single processor at the sustained rate of 175 MFLOPS. Approximately 55%, 20% and 20% of the computing time are spent for the implicit matrix operation, the numerical dissipation and the evaluation of viscous fluxes respectively. It is interesting to note that the LU-SGS scheme requires less computational work per iteration than some explicit schemes. Based on experience with INS3D-LU[21], an incompressible flow code which employs the LU-SGS scheme and achieved 1.2 GFLOPS using 8 processors,[36] the CENS3D is expected to perform very well on shared-memory multiple processors. The LU-SGS algorithm has outperformed the existing implicit schemes on a massively parallel computer such as the Connection Machine CM-2 in a recent study.[37]

In order to validate the new CENS3D code, calculations have been performed for a NACA64A010 wing. The thickness to chord ratio of the wing, whose aspect ratio is 4, has been modified to 10.6%. The experiment was conducted in the RAE 8 x 8 foot wind tunnel by Mabey et al.[38] The model was mounted on a fuselage-like body to displace it slightly from the wind tunnel wall and its boundary layer. However, no attempt has been made here to model the test section. A 151 x 39 x 39 C-H mesh (229,671 points) generated by Chaderjian[39] is used for the present calculation. Figure 2 shows a partial view of the computational grid. The freestream conditions are Mach 0.8, Reynolds number 2.4 x 10^6, and zero angle of attack. The algebraic turbulence model by Baldwin and Lomax[40] is employed for mathematical closure of the Reynolds-averaged Navier-Stokes equations. Original coefficients are used except C_{wk}, the coefficient for F_{wake} is set

to 1 instead of 0.25 as done in Ref. 39. The y^+ values at the first mesh cells which are adjacent to the wing surface near the midspan are about 2. The convergence history in Fig. 3 shows that the root-mean-squared residual of the continuity equation drops 3 orders of magnitude in about 340 iterations or 12 CPU minutes. Pressure contours are shown in Fig. 4. The computed pressure coefficients are compared with experimental data and the numerical solution of Chaderjian[39] in Figs. 5-7. His code uses a finite-difference discretization, artificial dissipation using blended second and fourth differences, a diagonalized ADI scheme, and the Baldwin-Lomax turbulence model. Figures 5-7 correspond to C_p comparisons at 50%, 77%, and 94% semi-span stations respectively. Values at the leading and trailing edges are not available for plotting because flow variables are located at cell centers. Overall agreements between the two numerical solutions are seen to be good despite the differences in numerical formulation. The slight discrepancy between the experimental data and the numerical solutions may be due to the effects of the fuselage-like body at the wing root and the wind tunnel wall which are not modeled in the numerical simulations.

For additional validation of the code, transonic flow calculations have been carried out for a ONERA M6 wing. A 289 x 50 x 44 C-H mesh (635,800 points) is used as a fine grid. The distance of the first grid point from the wing surface is 1.0 x 10^{-5} chord length of the root section. The freestream conditions are Mach 0.8395, Reynolds number 1.5 x 10^7, and 3.06° angle of attack. The Baldwin and Lomax turbulence model is used again for the attached flow simulation. The residual drops to 3 orders in about 380 iterations or 38 minutes of CPU time on the fine grid. In the present implementation, implicit viscous terms are not included to avoid the increase of computational work per iteration. To investigate the effect of this compromise on the convergence rate, a grid-convergence study has been performed. Fig. 8 shows the convergence histories on both fine grid and a 171 x 25 x 44 (188,100 points) coarse grid. Although the number of grid points to resolve the viscous boundary layer is doubled, the convergence is seen to be slowed by only twenty percent. Fig. 9 and Fig. 10 show a good agreement between experimental data[41] and the pressure coefficients at 44% and 65% semi-span stations computed on the fine grid.

Conclusions

A three-dimensional numerical method based on the LU-SGS implicit scheme in conjunction with the flux-limited dissipation model is developed for simulating viscous turbulent compressible flows. Good performance of the new testbed code is demonstrated on a Cray YMP computer. Despite its reasonably fast convergence, the LU-SGS scheme requires very low computational time per iteration. The present three-dimensional Navier-Stokes solution of a high Reynolds number flow using 636K grid points is obtained in 38 minutes.

References

1. Yoon, S. and Jameson, A., "Lower-Upper Symmetric-Gauss-Seidel Method for the Euler and Navier-Stokes Equations," AIAA Paper 87-0600, Jan. 1987. AIAA Journal, Vol. 26, Sep. 1988, pp. 1025-1026.

2. Yoon, S., "Numerical Solution of the Euler Equations by Implicit Schemes with Multiple Grids," MAE Report 1720-T, Princeton University, Sep. 1985.

3. Jameson, A. and Yoon, S., "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations," AIAA Journal, Vol. 25, July 1987, pp. 929-935.

4. Shuen, J.S. and Yoon, S., "A Numerical Study of Chemically Reacting Flows Using a Lower-Upper Symmetric Successive Overrelaxation Scheme," AIAA Journal, Vol. 27, Dec. 1989, pp. 1752-1760.

5. Choo, Y.K., Soh, W.Y., and Yoon, S., "Application of a Lower-Upper Implicit Scheme and an Interactive Grid Generation for Turbomachinery Flow Field Simulations," ASME Paper 89-GT-20, June 1989.

6. Yoon, S., and Jameson, A., "Lower-Upper Implicit Scheme for High-Speed Inlet Analysis," AIAA Journal, Vol. 25, Aug. 1987, pp. 1052-1053.

7. Rieger, H. and Jameson, A., "Solution of Steady Three-Dimensional Compressible Euler and Navier-Stokes Equations by an Implicit LU Scheme," AIAA Paper 88-0619, Jan. 1988.

8. Yu, S.T., Tsai, Y.L.P., and Shuen, J.S., " Three-Dimensional Calculation of Supersonic Reacting Flows Using an LU Scheme," AIAA Paper 89-0391, Jan. 1989.

9. Coirier, W.J., " High Speed Corner and Gap Seal Computations Using an LU-SGS Scheme," AIAA Paper 89-2669, July 1989.

10. Jameson, A., Schmidt, W., and Turkel, E., " Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes," AIAA Paper 81-1259, 1981.

11. Yoon, S. and Kwak, D., "Artificial Dissipation Models for Hypersonic External Flow," AIAA Paper 88-3708, July 1988.

12. Yoon, S. and Kwak, D., "Artificial Dissipation Models for Hypersonic Internal Flow," AIAA Paper 88-3277, July 1988.

13. Obayashi, S., "Numerical Simulation of Underexpanded Plumes Using Upwind Algorithms," AIAA Paper 88-4360-CP, Aug. 1988.

14. Chen, C.L., McCrosky, W.J., and Obayashi, S., " Numerical Solutions of Forward-Flight Rotor Flow Using an Upwind Method," AIAA Paper 89-1846, June 1989.

15. Loh, H.T. and Golafshani, M., "Computation of Viscous Chemically Reacting Flows in Hybrid Rocket Motors Using an Upwind LU-SSOR Scheme," AIAA Paper 90-1570, June 1990.

16. Yungster, S., "Numerical Study of Shock-Wave/Boundary Layer Interactions in Premixed Hydrogen-Air Hypersonic Flows," AIAA Paper 91-0413, Jan. 1991.

17. Imlay, S.T. and Eberhardt, S., " Nonequilibrium Thermo-Chemical Calculations Using a Diagonal Implicit Scheme," AIAA Paper 91-0468, Jan. 1991.

18. Park, C. and Yoon, S., "Calculation of Real-Gas Effects on Blunt-Body Trim Angles," AIAA Paper 89-0685, Jan. 1989.

19. Park, C. and Yoon, S., "A Fully-Coupled Implicit Method for Thermo-Chemical Nonequilibrium Air at Sub-Orbital Flight Speeds," AIAA Paper 89-1974, June 1989.

20. Park, C. and Yoon, S., "Calculation of Real Gas Effects on Airfoil Aerodynamic Characteristics," AIAA Paper 90-1712, June 1990.

21. Yoon, S., Kwak, D., and Chang, L., "LU-SGS Implicit Algorithm for Three-Dimensional Incompressible Navier-Stokes Equations with Source Term," AIAA Paper 89-1964-CP, June 1989.

22. Giles, M., Drela, M., and Thompkins, W.T., "Newton Solution of Direct and Inverse Transonic Euler Equations," AIAA Paper 85-1530-CP, 1985.

23. Beam, R. and Warming, R.F, "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations," AIAA Journal, Vol. 16, Apr. 1978, pp. 393-402.

24. Briley, W.R. and McDonald, H., "Solution of the Multidimensional Compressible Navier-Stokes Equations by a Generalized Implicit Method," Journal of Computational Physics, Vol. 24, No. 4, Aug. 1977.

25. Pulliam, T.H. and Chaussee, D.S., "A Diagonal Form of an Implicit Approximate Factorization Algorithm," Journal of Computational Physics, Vol. 39, 1981, pp. 347-363.

26. Obayashi, S. and Kuwahara, K., "LU Factorization of an Implicit Scheme for the Compressible Navier-Stokes Equations," Journal of Computational Physics, Vol. 63, Mar. 1986, pp. 157-167.

27. Steger, J.L. and Warming, R.F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite Difference Methods," Journal of Computational Physics, Vol. 40, No. 2, Apr. 1981, pp. 263-293.

28. Ying, S.X., Steger, J.L., Schiff, L.B., and Baganoff, D., "Numerical Simulation of Unsteady, Viscous, High Angle-of-Attack Flows Using a Partially Flux Split Algorithm," AIAA Paper 86-2179, 1986.

29. Rizk, Y.M., Chaussee, D.S., and Steger, J.L., "Numerical Simulation of the Hypersonic Flow Around Lifting Vehicles," NASA TM-89444, 1987.

30. Edwards, T.A. and Flores, J., "Toward a CFD Nose-to-Tail Capability: Hypersonic Unsteady Navier-Stokes Code Validation," AIAA Paper 89-1672, 1989.

31. Jameson, A. and Turkel, E., "Implicit Schemes and LU Decompositions," Mathematics of Computation, Vol. 37, No. 156, 1981, pp. 385-397.

32. Buning, P.G. and Steger, J.L., "Solution of the Two-Dimensional Euler Equations with Generalized Coordinate Transformation Using Flux Vector Splitting," AIAA Paper 82-0971, 1982.

33. Whitfield, D.L., "Implicit Upwind Finite Volume Scheme for the Three-Dimensional Euler Equations," Mississippi State University Report MSSU-EIRS-ASE-85-1, Sep. 1985.

34. Buratynski, E.K. and Caughey, D.A., "An Implicit LU Scheme for the Euler Equations Applied to Arbitrary Cascades," AIAA Paper 84-0167, 1984.

35. MacCormack, R.W., "Current Status of Numerical Solutions of the Navier-Stokes Equations," AIAA Paper 85-0032, 1985.

36. Fatoohi, R. and Yoon, S., " Multitasking the INS3D-LU Code on the Cray Y-MP," AIAA Paper 91-1581, June 1991.

37. Fatoohi, R., Private Communication.

38. Mabey, D.G., Welsh, B.L., and Pyne, C.R., "A Summary of Measurements of Steady and Oscillatory Pressures on a Rectangular Wing," The Aeronautical Journal of the Royal Aeronautical Society, Jan. 1988.

39. Chaderjian, N. M. and Guruswamy, G. P., "Unsteady Transonic Navier-Stokes Computations for an Oscillating Wing Using Single and Multiple Zones," AIAA Paper 90-0313, Jan. 1990.

40. Baldwin, B.S. and Lomax, H., " Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flow," AIAA Paper 78-0257, Jan. 1978.

41. Schmitt, V. and Charpin, F., " Pressure Distributions on the ONERA M6 Wing at Transonic Mach Numbers," AGARD AR-138-B1, 1979.

Parallel Computation of 3-D Navier-Stokes Flowfields for Supersonic Vehicles

by
James S. Ryan

This project involved development and testing of CFD tools for use on parallel computers. In the short term, this work supports development of High Speed Civil Transport (HSCT) designs as part of the High Performance Computing and Communications Program (HPCCP) Grand Challenges. The long-range goal is to enable teraflops-rate multidisciplinary optimization of aerospace vehicles. A more complete description of both the program and the technical results is given in the attached paper, James S. Ryan and Sisira Weeratunga, ``Parallel Computation of 3-D Navier-Stokes Flowfields for Supersonic Vehicles,'' AIAA Paper 93-0064, Reno, NV, January 1993.

Milestones

The following is the list of planned accomplishments from the proposal, along with the work done to satisfy each of them:

1. Rebuild the essential features of the serial CNS code around the parallel ARC3D algorithm developed by Sisira Weeratunga. This will satisfy the HPCCP milestone for June 1992.

The I/O routines required for use of the code were completed in the previous contract period. The Baldwin-Lomax model was parallelized, and include in the CFD code. A c-grid boundary condition was added to the code, for cases where the cut lies in a single processor.

Going beyond CNS capabilities, Weeratunga added Chimera-grid capabilities to the code, and I used this new feature to compute flow for a wing-body-nacelle case.

2. Validate the parallel CNS code using simple test cases which have analytical or experimental data available.

F Initial testing showed identical numerical behavior to the ARC3D algorithm on a Cray computer, so validation results from the Cray should be applicable here. The following cases have been computed to add confidence and demonstrate applicability to HSCT cases.

Flat plate boundary layer cases were used to test the laminar and turbulent capabilities. Results at Mach 2.0 match computational and analytical results well. A wing-body Euler calculation showed good agreement to available Cray results using the UPS space-marching code.

3. Demonstrate the success of CNS on the Intel iPSC/860 by solving an HSCT wing-body case. The first case will use a single zone grid, but multiple processors.

An Euler case completed as this contract period was beginning, satisfied this item. The geometry was a modern supersonic transport design. In addition, the Euler case was used to test the scalability of the code, and a fine-grid version was run to provide better validation. Results compared well with UPS results from the Cray Y-MP. Another single-zone case treated the same body with turbulent flow at a Reynolds number of 1 million based on body length.

4. Solve an HSCT wing-body case with multiple zones and a finer grid. This will meet the HPCCP milestone for January 1993.

The multiple-zone capability was tested by the addition of engine nacelles to two HSCT geometries. The first case run was one nacelle and the wing lower surface of a proprietary HSCT geometry. In order to generate results on a less sensitive (but still proprietary) geometry, generic nacelles were added to the existing wing-body grid. Overset gridding was used, adding only about 3% overhead relative to single-zone computations on the same grids.

5. Support development of an optimizing version of CNS.

As planned, this was a low-level effort, consisting mainly of helping others learn to use the Intel parallel computer effectively.

Other Work

In addition to the purely technical work, considerable effort was applied to disseminating results, and to exposing this work within the HPCC program. This resulted in the following presentations and contributions to presentations made by others:

November 1991:

- Provided a graphic representing my wing-body results and computational rates to Tom Edwards for use in a review for Ron Bailey. Bailey responded favorably to the results, which were possibly the first 3-D external flow calculations on the massively parallel machines. He suggested sending the results to Washington.

- Presented my results from tests of the Concurrent File System (CFS) on the iPSC/860 to a Parallel I/O Special Interest Group at Supercomputing '91 in Albuquerque, New Mexico. The presentation was well received by the Intel personnel and other researchers present.

December 1991:

- Completed production of a video explaining my CFD work on the Intel computer, and its place in HPCCP. The content was directed at interested non-technical viewers, such as congressmen who would be shown the video as part of the budgeting process. The video went to Washington with Ken Stevens for review within NASA. Portions were included in a professionally produced video called "Grand Challenges 1993."

January 1992:

- Presented a review of the Branch's work on the HPCCP HSCT Grand Challenge to Lee Holcomb of NASA Headquarters.

February 1992:

- Provided print and transparency graphics to Terry Holst, Ken Stevens, and Tom Lasinski. These HPCCP-related graphics depicted my HSCT test-case solution on the Intel iPSC/860.

- Provided copies of my CFS I/O paper to Intel employees at Ames and at Caltech.

May 1992:

- Provided graphics of wing-body Euler results to Jolen Flores, with additional information for use by Paul Kutler.

- Presented recent results to the local CAS applications group, and prepared slides for a more extensive presentation in Cleveland next month.

June 1992:

- Attended the Computational Aerosciences Industry Briefing at Cleveland, Ohio. Presented a 20 minute (plus questions) talk on recent work in the use of parallel computers for Navier-Stokes CFD computations.

August 1992:

- Presented a talk entitled ``Parallel Navier-Stokes Computation of Supersonic Vehicle Flowfields,'' at the NASA Computational Aerosciences Conference, August 18-20, 1992. A compendium of abstracts was published.

- Prepared materials for inclusion in the HPCCP annual report being prepared by Lee Holcomb at NASA headquarters.

October 1992:

- Sent out a 427 form, proposing to present the content of AIAA Paper 93-0064 at the "Parallel CFD '93" Conference in Paris, France, in May of 1993. This 427 will probably be rejected, on grounds of economic sensitivity of the technology.

AIAA 93-0064
Parallel Computation of 3-D Navier-Stokes Flowfields for Supersonic Vehicles

J. S. Ryan
MCAT Institute

S. K. Weeratunga
Computer Sciences Corporation

31st Aerospace Sciences Meeting & Exhibit
January 11-14, 1993 / Reno, NV

PARALLEL COMPUTATION OF 3-D NAVIER-STOKES FLOWFIELDS FOR SUPERSONIC VEHICLES

James S. Ryan* and Sisira Weeratunga†
NASA Ames Research Center
Moffett Field, California

Abstract

Multidisciplinary design optimization of aircraft will require unprecedented capabilities of both analysis software and computer hardware. The speed and accuracy of the analysis will depend heavily on the computational fluid dynamics (CFD) module which is used. A new CFD module has been developed to combine the robust accuracy of conventional codes with the ability to run on parallel architectures. This is achieved by parallelizing the ARC3D algorithm, a central-differenced Navier-Stokes method, on the Intel iPSC/860. The computed solutions are identical to those from conventional machines. Computational speed on 64 processors is comparable to the rate on one Cray Y-MP processor, and will increase as new generations of parallel computers become available.

Objective and Motivation

New aerospace vehicles must meet higher standards than ever before, in order to provide technical and economic advantages over older generations of aircraft. They must offer low maintenance costs and economical fuel consumption. Lower limits will be enforced for pollutant emissions and airport noise. On many routes, supersonic flight may provide a competitive advantage, leading to interest in a High Speed Civil Transport (HSCT). For such a transport aircraft, supersonic flight must be combined with environmentally acceptable sonic boom levels. Additionally, efficient subsonic cruise must be possible, to ensure access of the HSCT to areas where supersonic flight may be prohibited. In order to design such an aircraft, it is no longer adequate to consider external aerodynamics, propulsion, structures, and controls in isolation. The simulations used to evaluate a design must take into account several of these disciplines for each flight regime, from takeoff and landing, to transonic operation, to supersonic cruise. Numerical optimizers will use a series of such simulations to find optimal values for large sets of design parameters.

These multidisciplinary simulations will require computational power beyond the reach of traditional vector supercomputer architectures. The High Performance Computing and Communications Program (HPCCP) has selected the HSCT as one of several Grand Challenges, which will be used to explore the power of parallel computers, while simultaneously contributing to the solution of problems of scientific, technical, and economic importance. As a step toward multidisciplinary computation on highly parallel computers, a parallel CFD code has been developed. This CFD module is designed for integration with modules providing analysis capabilities for structures, propulsion, and other disciplines, to create a complete multidisciplinary design tool.

This project also provides feedback to the developers of parallel architectures, hardware, operating systems, and compilers. The practical experience of building aerospace design tools on parallel computers can encourage and guide the development of the next generation of parallel hardware and software.

Technical Approach

The present work focuses on the development of a versatile computational fluid dynamics module for High Speed Civil Transport (HSCT) flow fields. Buning's "Overflow[1]" implementation of ARC3D[2] serves as the basis for the parallel version described in the next section. By basing the flow solver on existing, well-proven serial algorithms, the uncertainties surrounding a totally new algorithm are avoided. The new parallel version of ARC3D gives results which are identical, aside from roundoff error, to those from Cray versions.

Complex vehicle designs are often difficult to grid in a single zone. Building a usable single-zone structured grid around a wing-body with nacelles is difficult. With the addition of control surfaces and an empennage, the problem becomes practically impossible. This problem is alleviated by gridding components of the aircraft separately, either in a patched or overset grid approach. The present code includes both of these

* MCAT Institute, Member AIAA
† Computer Sciences Corporation, Member AIAA

capabilities. The patched grid approach is based on the successful methods used in such codes as TNS[3] and CNS,[4] although the parallel implementation required complete re-coding. The overset, or Chimera[5], approach is based on the "Overflow" code, and uses input from either Pegsus 4.0[6], or Meakin's DCF3D[7] code. Each zone is built so that its outermost points correspond to interior regions of the adjacent zones. The gridding of each zone is independent, except in the overlap areas of patched grids. The implementation of the Chimera grids is described in a later section.

In addition to the usual physical boundary conditions, "boundary condition coupling" will serve to integrate the CFD module with other disciplines. For example, pressures from the CFD module can provide input to a structures code, which will feed back a modified surface shape to the CFD code. This surface shape requires modification of the flowfield grid in the CFD problem. For unsteady problems, the surface velocity becomes one of the boundary conditions for the next flow solver iteration. Each discipline can provide updated boundary conditions for the others, as often as necessary to provide time accuracy in each part of the problem.

Algorithm Implementation

The Intel iPSC/860 System

The Intel iPSC/860 system is an aggregate of interconnected processor nodes. Each processor, or computational node, consists of an Intel i860 microprocessor with memory and inter-node communication components. The iPSC/860 at NASA Ames Research Center consists of 128 such nodes, each with 8 Mbytes of memory. The i860 is a 40 MHz reduced instruction set (RISC) microprocessor chip with a theoretical peak execution rate of 32 MIPS integer performance and 60 Mflops 64-bit floating-point performance. The 128 node iPSC/860 delivers an aggregate peak performance of over 7 Gflops on 64-bit data and supports a total of one Gbyte of random access memory. These peak performance rates are based on ideal conditions with regard to the mix of instructions, cache utilization, pipelining, data alignment etc. Such optimal conditions do not occur in practical applications such as CFD.

The processors in the 128 node iPSC/860 are interconnected by a 7-dimensional hypercube communication network. Each computational node interfaces with the network through a dedicated communication processor called the Direct Connect Module (DCM). The DCM can supervise up to 8 full duplex serial channels simultaneously with a peak data transfer rate of 2.8 Mbytes per second per channel. It also provides hardware by-pass switching (i.e., worm-hole routing) for every node in the system. As a result, messages can pass equally quickly between adjacent nodes and nodes at the opposite corners of the interconnection network, provided there is no link contention. Thus, it effectively emulates a fully connected network, with very little penalty for non-local communication.

Attached to the communication network are 10 I/O nodes, each of which is an Intel 80386 processor with approximately 700 Mbytes of disk space. These I/O nodes form the Concurrent File System (CFS) with a total capacity of 7 Gbytes. The disks in the CFS are directly accessible to the computational nodes over the interconnection network. The peak data transfer rate between a single computational node and the CFS is about 1.5 Mbytes per second. This translates into a peak transfer rate of approximately 15 Mb/sec. However, the actual transfer rates realized in practical computations are much lower due to contention for I/O nodes, network congestion and inefficient cache utilization.

The iPSC/860 is controlled by an intermediate host computer, referred to as the System Resource Manager (SRM). The SRM serves as the machine's interface to the outside world by providing such functions as system resource management and external network access. Each of the computational nodes in the iPSC/860 system runs a simplified operating system kernel known as NX/2 that supervises process execution and supports buffered, queued message passing over the interconnection network with other computational nodes, I/O nodes and the SRM.

In distributed memory machines such as the iPSC/860, there is no globally shared, directly addressable memory. Instead, each processor has a private address space in a private memory. As a result, each processor runs its own version of the program and data is communicated between processors by means of a "send-receive" protocol explicitly coded in each program. In addition to the sharing of information, this mechanism is also the primary means of synchronization between processors. Consequently, computation on distributed memory machines can be visualized as a system of communicating sequential processes. The messages exchanged have relatively high communication latencies (approximately 65-150 microseconds) and low communication bandwidths. Hence, there is a significant performance penalty for moving data between processors frequently and/or in large quantities.

Parallel Implementation Considerations

The goal of the parallel implementation is the extraction of maximum parallelism to minimize the execution time of the application on a given number of processors. However, there are several different types of overheads associated with a parallel implementation. These include communication overhead, data dependency delays, load imbalance, arithmetic overhead, and memory overhead. Here, the arithmetic and memory

overheads refer to the extra arithmetic operations and memory needed by the parallel implementation when compared with the best equivalent serial implementation. While the first four types of overheads lead to performance degradation, the memory overhead may limit the size of the problem that can be run on a given system. In practice, minimizing all these overheads simultaneously is difficult. Thus, most practical parallel implementations require the developer to make compromises with regard to different types of overheads with the overall goal of achieving a near-minimum execution time, subject to a reasonable programming effort.

A given application consists of several different, independent algorithmic phases that must be performed in a prescribed sequential order. In addition, the degree of parallelism and the type of data dependencies associated with each of these subtasks can vary widely. Here the degree of parallelism refers to the order of magnitude of the number of finest granularity concurrent subtasks.

The version of ARC3D implemented in this study is the diagonal form of the Beam and Warming implicit approximate factorization algorithm for the solution of the Reynolds-averaged Navier-Stokes equations[2]. A single time step of this implicit time integration scheme can be considered to comprise six different types of subtasks: (a) enforcement of boundary conditions, (b) formation of right hand side vector (RHS) involving Euler, viscous and smoothing terms, (c) block-diagonal matrix-vector multiplications involving (5x5) elemental similarity transformation matrices, (d) formation of scalar pentadiagonal systems of equations involving Euler, viscous and smoothing terms, (e) solution of multiple, independent systems of scalar pentadiagonal equations and (f) solution update. In the following section, we describe each of these tasks with respect to their impact on the parallel implementation. In this discussion, N refers to a typical dimension of the computational domain.

The degree of extractable parallelism associated with subtask (a) is $O(N^2)$. In addition, since the enforcement of boundary conditions is done only at the boundaries of the computational domain, the distribution of load is not homogeneous. The severity of this load imbalance is dependent on the mix of boundary conditions used in the application. While most boundary conditions have only local spatial data dependencies, there are others that contain non-local spatial data dependencies. Examples of such boundary conditions are C-grid flow-through conditions, periodic/axis conditions and evaluation of surface pressure based on normal momentum equations. Enforcement of such non-local boundary conditions may require inter-processor communication and could occupy a significant fraction of run time. The only mitigating factor is that in most practical problems, the ratio of boundary to interior points is small.

The subtasks of type (b), (c), (d) and (f) are typified by $O(N^3)$ degree of extractable parallelism with homogeneous distribution of the computational load. In addition, the spatial data dependencies associated with these tasks are highly localized. They are either nearest or next-to-nearest neighbor for second-order spatial accuracy.

The sequentially optimum algorithm for subtask (e) involves second-order recursion. This eliminates the possibility of extracting any parallelism in the solution of a single, scalar pentadiagonal system. Therefore, to extract any concurrency in the solution of such a system requires that the sequential algorithm be replaced by one with exploitable parallelism. Most such algorithms incur substantial arithmetic and communication overheads and may not reduce the execution time significantly. However, subtask (e) involves the solution of multiple, independent systems of scalar pentadiagonal equations in each coordinate direction, with the multiplicity being $O(N^2)$. This exposes an easily extractable $O(N^2)$ degree of parallelism. The degree of extractable parallelism can be further enhanced by using the concept of pipelined data parallel computation. This is one of the approaches used in this study.

Data Partitioning in ARC3D

Analysis of the extractable parallelism of various subtasks of ARC3D in the previous section indicates that the finest level of subtask granularity for most computations is at the grid-point level. The exception is for the subtasks of type (e), where the finest level of granularity is at the level of a group of grid points in a given coordinate direction. Therefore, it is natural to decompose the data space of ARC3D at the level of group of grid points in each coordinate direction. This is referred to as grid partitioning. The idea is to map the subdomains (i.e., processes) so created onto the processors in such a way that the distribution of grid points leads to a nearly balanced load of computation and communication. It is also desirable to maintain the spatial locality of the grid structure in order to minimize the amount of communication.

In the case of structured grids, as used in ARC3D, this is easily achieved by partitioning the computational domain into logically congruent, nearly equal-sized rectangular parallelepiped-shaped subdomains. Since the subgrids created by this partitioning are themselves structured, the nodal programs written for the individual processors will bear a close resemblance to the program structure of a sequential implementation. The parallel implementations based on such partitioning schemes posses the following characteristics: (1) the underlying numerical algorithms are not changed, i.e., the parallel implementation give exactly the same results as the sequential version; (2) processors are programmed homogeneously, i.e., the Single Program, Multiple Data (SPMD) model is used; (3)

implementations are independent of the topology of the interconnection network and the number of computational nodes (provided the local memory capacity is sufficient for a problem of a given size); (4) communication patterns for data exchange among processors are simplified; (5) computational and communication load are equally distributed among the processors for tasks with homogeneous, grid-point level parallelism.

In this study, one grid subdomain is assigned to each of the processors. Such a partitioning scheme is referred to as a uni-partitioning scheme. The simplest and most commonly used structured grid partitioning scheme slices the computational domain along planes normal to each of the coordinate directions. As a result, the maximum number of partitions in a given coordinate direction is limited to the number of grid points in that direction. When the computational domain is sliced only along one coordinate direction, it is referred to as a 1-D partitioning. Similarly, slicing the grid in two or three coordinate directions gives a 2-D or 3-D partitioning scheme, respectively.

The highest dimensionality of the partitioning scheme that can be used for a given grid-oriented algorithm depends on the degree of extractable parallelism of that algorithm. The optimum partitioning depends on the algorithm's computational and communication requirements, machine architectural features, and the number of grid points in each coordinate direction. For a problem of fixed size, use of higher dimensional partitioning, if feasible, facilitates the use of a larger number of processors.

Implementation Details of ARC3D

We have implemented ARC3D on the iPSC/860 by using 3-D uni-partitioning of the computational domain. However, 1-D and 2-D uni-partitionings are subsets of this implementation. Each subdomain is assigned to a computational node of the iPSC/860. This assignment can be either algebraic (i.e., i-th subdomain to the i-th processor) or it can be in such a way that neighboring subdomains are mapped onto processors that are directly connected in the hypercube communication topology. Such a mapping is feasible for all three types of partitionings because the hypercube topology allows the embedding of rings, 2-D and 3-D meshes through the binary reflected Gray code. One advantage of a such an assignment scheme over a naive assignment is that it tends to minimize the distances traveled by the messages and the potential for network link contention, at least in data exchanges involving neighboring subdomains. However, our experimental performance data do not show any significant advantage for this type of process-to-processor mapping scheme. This appears to partially substantiate Intel's claims regarding DCM's ability to mimic the appearance of a fully-connected network.

Under this statically determined uni-partitioning scheme, the solution variables held in each subdomain are computed by their associated computational node. During the RHS evaluation, interior faces of a subdomain require solution values held by the adjacent subdomains. A given subdomain may require such data from up to six other subdomains. Instead of exchanging these values exactly at the instant they are required, the data are stored in so-called overlap areas by allocating storage for one extra grid point in each of the six directions of the subdomain computational grid. This allows for the exchange of internal boundary data by processors holding adjacent subdomains via a few, relatively long messages. As a result, the cost of latency associated with message passing is minimized, resulting in reduced communication overhead. However, the allocation of storage for such overlap areas and the need for using equally long message buffers during the data exchange process results in substantial memory overhead. The introduction of such overlap areas leads to an implementation equivalent to the sequential one, since a strict coherency is maintained between data in the overlap areas and those on the subdomain internal boundaries. At first glance it appears as if the presence of fourth-difference dissipation terms would require two extra grid points in each of the six directions for the overlap areas. However, by exchanging the second-differences during the computation of smoothing terms, the need for an extra layer of grid points in the overlap areas is avoided. The data dependency delay overhead in these computations is limited to that associated with the exchange of data in the overlap areas. The primary reason for such delays is the load imbalances associated with subtasks of type (a) and (e). In addition, there is an arithmetic overhead, due to the redundant computation of various flux data in the overlap areas as well as a communication overhead due to exchange of data in those areas.

As mentioned earlier, the solution of the scalar pentadiagonal systems induces global data dependencies. There are a variety of concurrent algorithms available for this task. We have considered three such algorithms: (1) Complete-exchange based implementations (CE-GE), (2) Pipelined Gaussian elimination (PGE), and (3) Sub-structured Gaussian elimination followed by solution of the reduced system via balanced odd-even cyclic reduction (SGE-BCR). The complete exchange or global transpose based implementations are limited to $O(N^2)$ degree of extractable parallelism but contain no arithmetic overhead. Also, such implementations are typically associated with high memory and communication overhead. The interprocessor communication is characterized by a relatively small number of messages of length $O(N^3)$. The pipelined (both one-way and two-way) Gaussian elimination algorithms, while exhibiting $O(N^3)$ degree of

parallelism and no arithmetic overhead, suffer from high memory overhead and processor idling during pipeline filling and draining. In addition, they are characterized by a large number of relatively short messages that may lead to inefficiencies on systems with high message latencies. In contrast, the sub-structured Gaussian elimination based algorithms exhibit $O(N^3)$ degree of readily extractable parallelism, but suffer from relatively high arithmetic and memory overhead.

Under the uni-partitioning schemes, subdomains containing external boundary faces are held only by a subset of the processors. Therefore task (a) is processed only by those nodes holding those faces, while others may be idle. The severity of this load imbalance is short-lived for most common types of boundary conditions needed in practical flow simulations.

The Baldwin-Lomax turbulence model[8] is implemented in the current code. This model requires searching in the wall-normal direction for the maxima of certain flow parameters. In the parallel version this often requires searching across several processors. The model finds local maxima in each processor and compares values from all applicable processors, in order to give eddy viscosity values which are unaffected by the partitioning of the grid. The searches are performed largely in parallel, so that the computational time consumed is minimized. In flat plate test cases, searching only the points assigned to one processor added 5 percent to the total computational time. Searching through 4 processors in the wall-normal direction added only 3 percent more time.

Implementation of Composite Grid Schemes

The overset grids used in the Chimera approach result in the embedding of both outer boundaries and solid body regions of one grid within the computational domains of other grids. The embedding of the solid body regions requires that certain grid points be blanked out within some neighborhood of the solid body region. These points are referred to as hole points. The grid points that lie in the fringes of this blanked-out region form an artificial interior boundary and serve to impose the effect of the embedded solid body region upon the grid. Consequently, the inter-grid boundaries of a composite grid are formed by the union of the embedded outer boundaries of the minor grids and the artificial interior boundaries of the blanked-out regions. In overset grid schemes, the effect of one grid is imposed upon the other by interpolating intergrid boundary data between them. In practice, this process is carried out at the end of each time step on each grid and is referred to as intergrid communication.

The flow field data needed to update the intergrid boundary points is interpolated from the solutions in the neighboring grids. Most interpolation schemes seek data from the nearest hexahedral computational cell in the overlap region. Such cells are referred to as donor cells. Therefore, to successfully carry out the intergrid communication process requires the identification of three types of grid points in all component grids: the hole points, the intergrid boundary points, and the donor cells. Currently, this information is provided as input by either Pegsus or DCF3D in a preprocessing step.

On conventional supercomputers, each component grid of the composite grid is generally treated sequentially, while the other components reside in a secondary storage device such as the SSD on Cray Y-MP. The iPSC/860 implementation of the overset grid scheme is based on the zonal decomposition approach. Interzone communication is accomplished through the inter-cube communication facility developed by Barszcz[9]. The zonal decomposition exploits the functional parallelism among multiple overlapping grids, and the data parallelism within each individual grid. As a result, all component grids are computed concurrently on different groups of processors with independent spatial data decomposition within each grid. The data partitioning is carried out in a manner that optimizes the performance of the parallelized implicit flow solver for each grid. The number of processors assigned to each component grid is decided on the basis of the computational load associated with the flow solver used for that grid. Given a fixed number of processors, this approach allows good static load balancing across the clusters of processors involved in the flow solver phase.

The intergrid data interpolation and communication is done concurrently, through a loosely synchronous approach. At the end of a time step, processors holding donor-cells in each component grid send the interpolated flow field data to the appropriate processors of the other component grids. Each processor proceeds to the computations of the next time step of the flow solver as soon as its intergrid communication phase is completed. A distributed intergrid communication data structure is used to minimize the memory overhead. No attempt is made to equidistribute the intergrid boundary points or the donor cells associated with each grid. Thus, during intergrid communication, there are likely to be significant load imbalances within each group of processors as well as across the groups of processors. This load imbalance is tolerable, as long as the time spent on intergrid communication process is a relatively small fraction of the time required to compute a single time step of the flow solver. The timing data for the composite grid configurations investigated so far indicate that the intergrid communication overhead is less than 3%.

I/O Considerations

Input and output of grid and solution files are usually minor considerations on conventional computers. Methods for I/O are straightforward, and practically

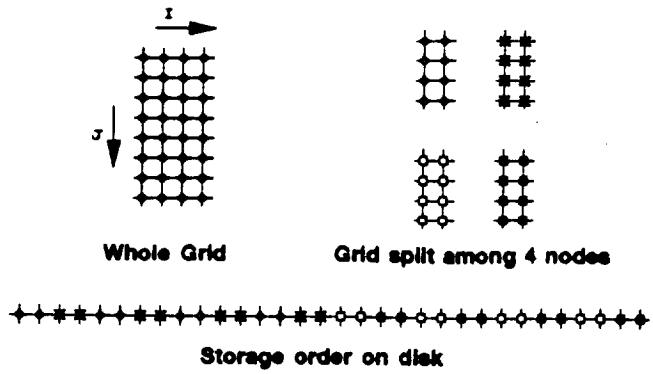

Figure 1. Schematic of a 32 point grid distributed among 4 nodes. To reassemble the grid in a single file requires 16 writes to the CFS.

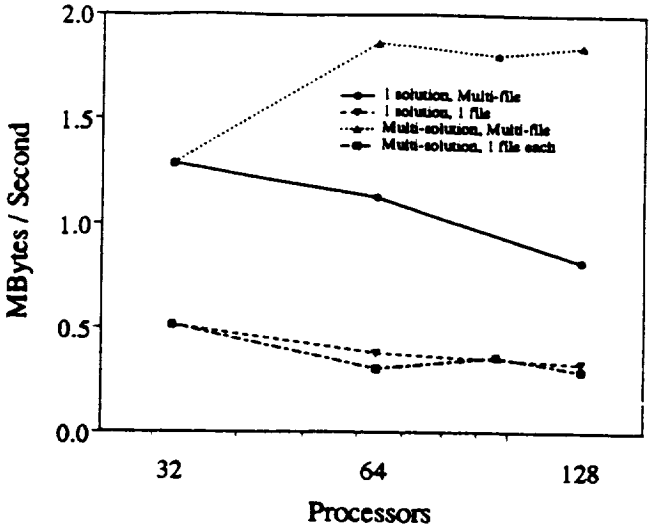

Figure 2. Transfer rates for 402,000-point CFD solution files. The solutions are output either one file per solution, or one file per processor (multi-file). Multi-solution indicates that several solutions were output, each from a separate group of 32 processors.

no CPU time is consumed, since idle processors become available to other users. On the iPSC/860, processors which are waiting for I/O are still dedicated to the calling process, so any idle CPU time is lost. Also, the parallel aspects of I/O between multiple processors and multiple disks add to the complexity of the operation. During processing, the data representing the flow solution is distributed across many processors. When that data is written to the disks of the Concurrent File System (CFS), it is often useful to store the data as a single array of values, rather than in pieces which correspond to each processor. This allows the solution to be used for restart on any number of processors, and allows postprocessing on workstations without re-ordering the data. In general, this requires each processor to write small amounts of data to many separate locations on the disks, to order the data correctly. This is illustrated for a very small grid in Fig. 1. These numerous write operations result in inefficient use of the cacheing capability of the I/O subsystem, and contribute to delays due to contention for the I/O nodes.

In early testing of the I/O routines, up to 5 Mb/sec was achieved from 16 processors to a single output file. For larger numbers of processors, the rates actually drop. Several tests were made with a 402,000-point grid, which requires a minimum of 32 processors. Additional processors were included either by distributing the single grid over more processors, or by running multiple 32-processor cases in parallel. Solution files were written in two ways: either in a single file as described above, or as separate files containing the data from each processor. The multiple-file form of output is faster, and is used when a solution will be restarted on the same number of processors. The results are summarized in Fig. 2, which shows that the combined I/O rate from all processors never exceeds 2 Mb/sec for these cases. The transfer of a single-file CFD solution from the processors to CFS files requires from 4 to 26 times as long as an iteration of the flow solver. Solution output to separate files from each processor is somewhat faster, requiring from 1 to 8 times as long as a solver iteration.

Data transfer rates from the processors to the CFS are acceptable for steady state problems, which run hundreds of iterations before a solution must be stored. For unsteady problems, solutions must be stored frequently and I/O will consume a substantial fraction of the total CPU time for the problem. As these problems become more common, and as the computational speed of parallel computers increases, the I/O subsystems will have to improve rapidly.

Computational Results

The new parallel code was tested on a simple square-duct case, and found to give identical results to the serial version of the algorithm. Since the code behaves identically, validation work done with the ARC3D algorithm on serial machines is applicable to the new code as well. Several solutions produced with the new parallel code serve to add confidence in the parallel implementation, and to demonstrate the applicability of the code to the High Speed Civil Transport (HSCT). After the test cases are described, performance results are given, along with an evaluation of the current levels of performance.

The first results described are validation cases, for which there are some analytical or numerical results available for comparison. These include laminar and turbulent flat-plate boundary layers, and an Euler computation about a wing-body. Additional demonstration calculations include a Navier-Stokes solution about the wing-body, and a multiple zone calculation of the wing-body with generic engine nacelles added.

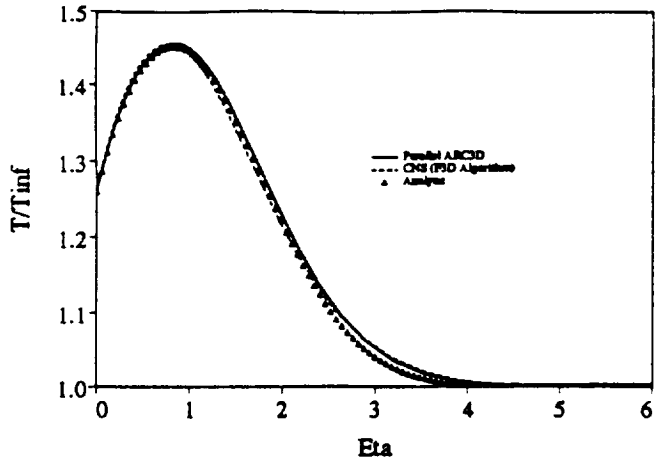

Figure 3. Temperature in the laminar boundary Layer 1 inch from the leading edge of a flat plate at $Re = 159,900$, $M_\infty = 2.5$, $T_\infty = 216.5K$, $T_{wall} = 273K$, $Pr = 1.0$

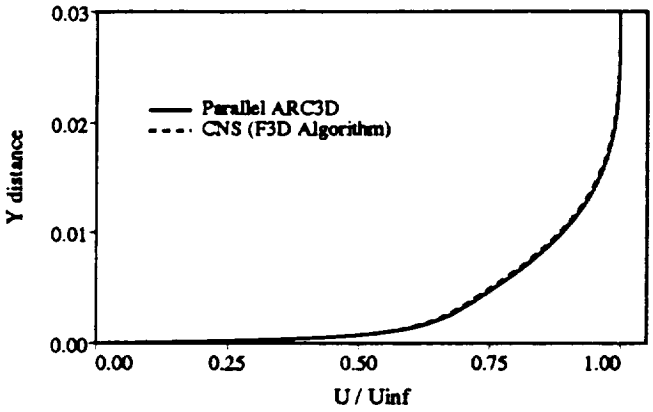

Figure 4. Turbulent boundary layer 1 meter from the leading edge of a flat plate at $Re = 1,000,000$, $M_\infty = 2.0$, $T_\infty = 275K$, $T_{wall} = 370K$, $Pr = 0.72$

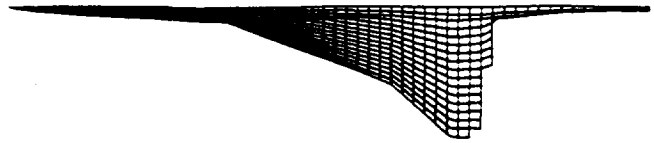

Figure 5. Upper surface grid on the wing-body. Only half of the points are shown in each direction.

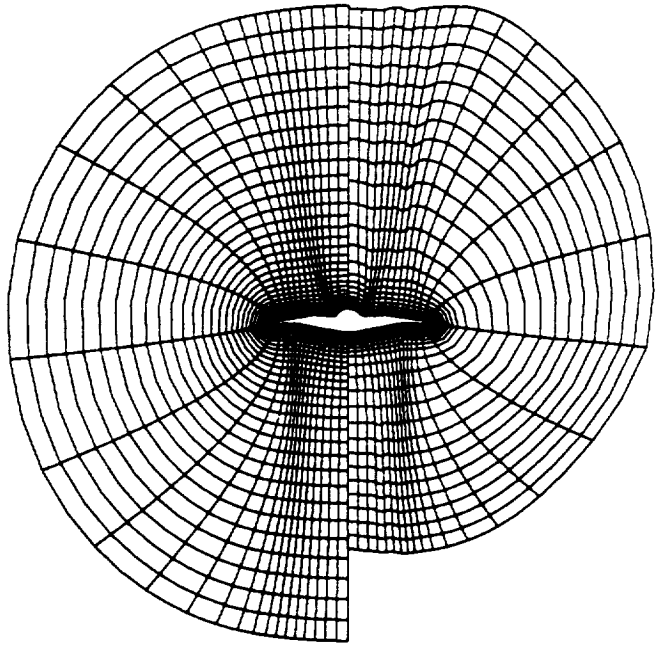

Figure 6. A crossflow plane of the volume grid at about 60 percent of body length. The Euler grid is on the left. The grid on the right has been modified to improve boundary layer resolution for turbulent cases. Only half of the points are shown in each direction.

Validation Cases

Two supersonic flat plate boundary layer cases show excellent agreement with serial codes. Fig. 3 shows the temperature profile in a laminar boundary layer over an isothermal flat plate. An analytical solution is plotted, as well as the results from the F3D[10] flux-split algorithm, which was run on a Cray Y-MP. The points resolving the boundary layer in each case extended across several processors in both the streamwise and wall-normal directions, providing an example of how the flow solver and boundary layer model are unaffected by processor boundaries. A boundary layer profile for the turbulent case is shown in Fig. 4. It compares well with F3D results from the Cray.

The first three-dimensional test case was an Euler solution about a modern HSCT wing-body. Nearly one-half million points were used in a 67x60x112 grid. The grid was generated in crossflow planes, so that a parabolized code, UPS[11], could easily be used for comparison. Each 67x60 point crossflow plane was a C-mesh covering half of the wing-body, plus one reflected plane. Fig. 5 shows the surface grid in a planform view, and Fig. 6 shows a crossflow plane of the volume grid. In order to distribute points among 32 processors, the grid was divided into 8 partitions in the streamwise direction, and 4 partitions in the body-normal direction. No fewer than 32 nodes could be used for this case, due to memory limitations.

The case was run with a freestream Mach number of 2.1, and an angle of attack of 4.75°. This is approximately the angle of attack for maximum lift-to-drag ratio. The converged solution was compared to the UPS results. Surface pressures, such as the centerline pressures shown in Fig. 7, compare well. The differences between the solutions are primarily due to differences in the way the two codes resolve the flow. The UPS code adds many intermediate planes in the streamwise direction, enhancing resolution, but introducing some differences due to interpolation.

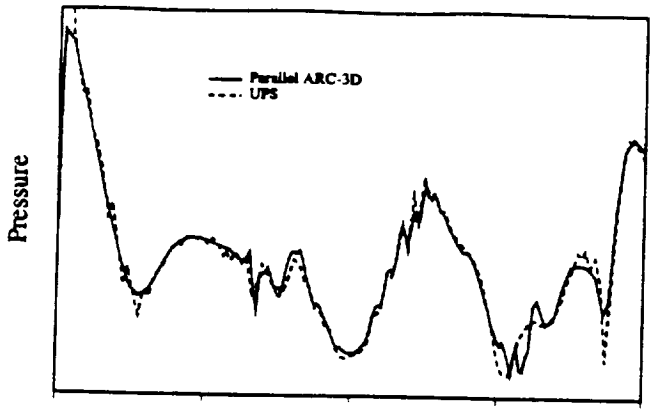

Figure 7. Pressures on the wing-body centerline.

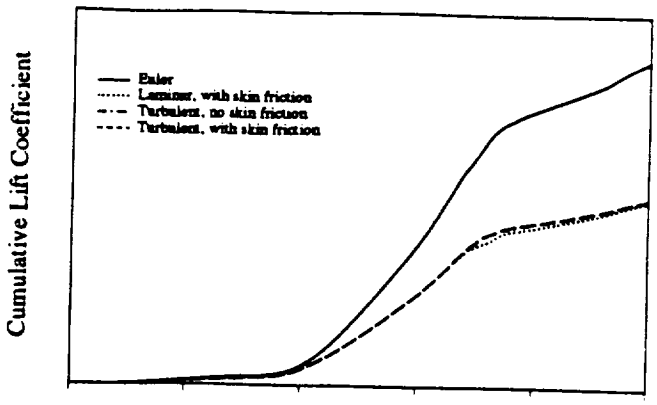

Figure 8. Lift coefficient results from Euler, laminar and turbulent cases on the wing-body. $M_\infty = 2.1, \alpha = 4.75°, Re = 1,000,000$

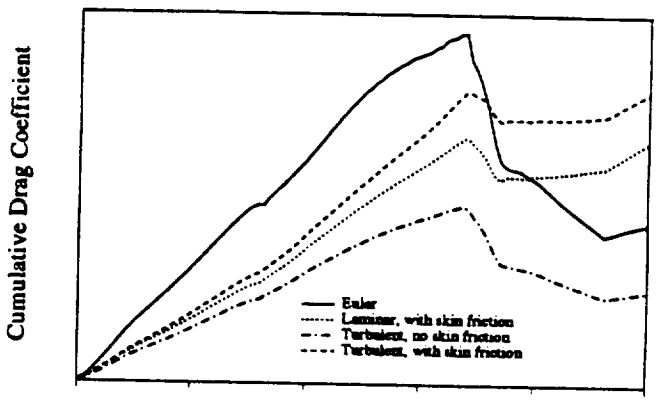

Figure 9. Drag coefficient results from Euler, laminar and turbulent cases on the wing-body. $M_\infty = 2.1, \alpha = 4.75°, Re = 1,000,000$

Demonstration Calculations

The wing-body has been solved as a Navier-Stokes calculation, with the Baldwin-Lomax turbulence model. This case demonstrates the Navier-Stokes capability, but does not validate it, since only Euler solutions were used with this geometry on serial computers.

The case was run at the same Mach number and angle of attack as the Euler case, and a Reynolds number based on body length of one million. In actual flight, the vehicle would have a Reynolds number on the order of 5.0×10^8 per meter. The lower number used here is reasonable for wind tunnel models, and allows the number of grid points to be kept small. A very simple grid adaption approach was used to modify the Euler grid to give a Y-plus of about 0.5 everywhere on the body. Y-plus is defined here as $y^+ = (\rho U_t \Delta y/\mu)$, where U_t is the flow speed tangent to the body, and Δy is the normal distance from the wall to the adjacent grid point. The grid was also modified to move the outer boundary inward, shifting unneeded points from outside the shock into the active flowfield. The adaption was repeated twice, giving an improved grid with negligible computational cost. The grid can be seen on the right side of Fig. 6.

Lift and drag results for the wing-body are shown in Figs. 8 and 9. The lift-to-drag ratio for the turbulent Navier-Stokes case is about three times lower than for the Euler case. The difference between the two is exaggerated by the low Reynolds number used in the test case, and in fact the turbulent case has the same lift-to-drag ratio as the Euler case if skin friction drag is ignored.

The final demonstration calculation is based on the Euler wing-body case. Grids for two generic engine nacelles were generated, and placed under the wing-body to demonstrate the overset-grid capability. Each nacelle was treated with two grids: one for the exterior, and another to allow flow through the interior. The two grids about each nacelle exchange information with each other and with the wing-body grid, as shown for example in Fig. 10. The planes shown are neither flat nor coincident, but they are close enough to serve as a 2-D illustration of the Chimera grid scheme, which is fully three dimensional. The planes shown are upstream of the nacelle, so no points are cut out of the wing-body grid at that point. The nacelle lip has zero thickness, and there is no diverter in this calculation.

The convergence rate of the five-zone wing-body-nacelle computation was nearly the same as for the wing-body alone. Since six additional processors handled computations for the nacelle grids, the time per iteration increased by only about 2.5 percent, which represents the cost of the Chimera interpolation and information exchange. The lift increment due to the nacelles was calculated, but proved to be negligible for this case. The changes in pressure on the wing lower surface are shown in Fig. 11.

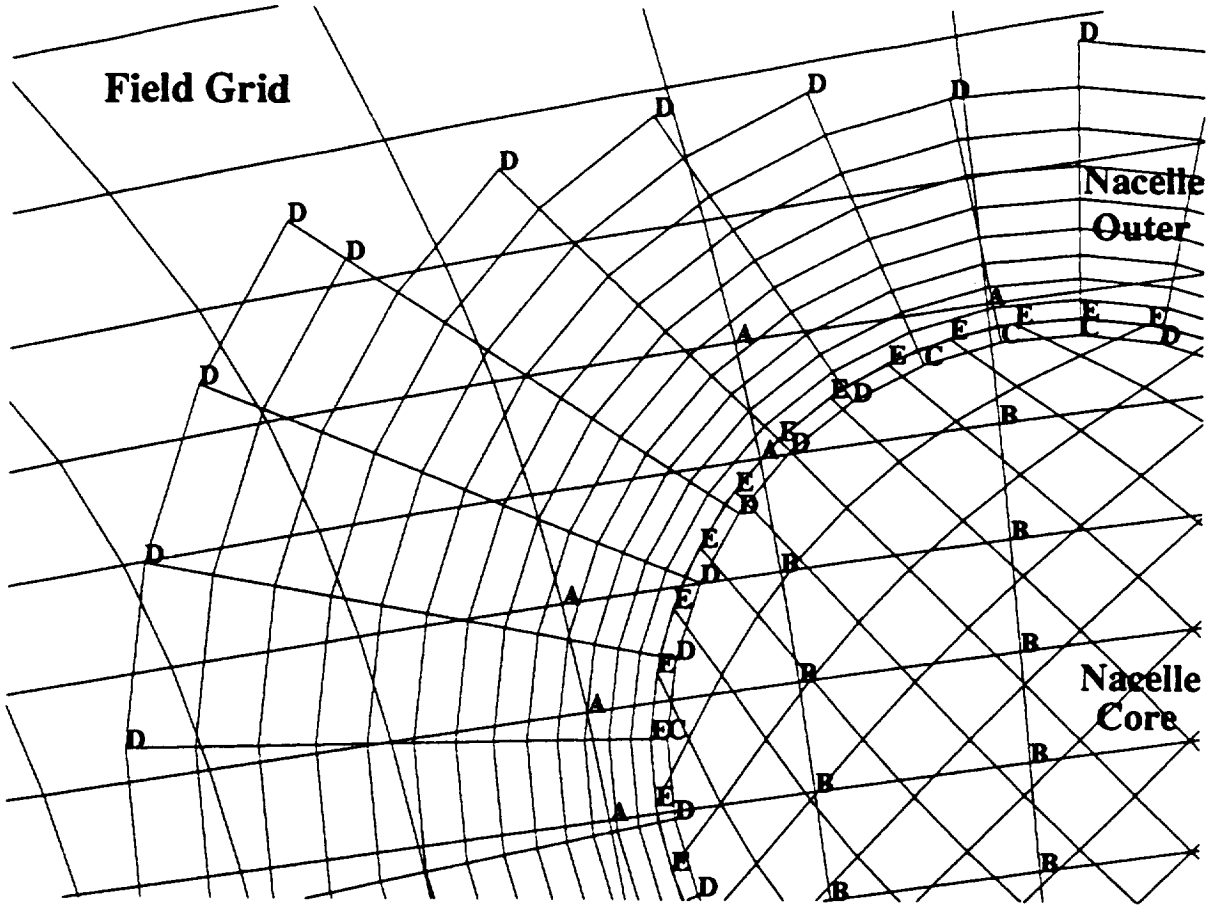

Figure 10. Grid interfaces just upstream of the inboard nacelle. A: Wing-body field grid receives information from nacelle outer grid; B: Field grid from nacelle core; C: Outer from core; D: Outer from field; E: Core from outer

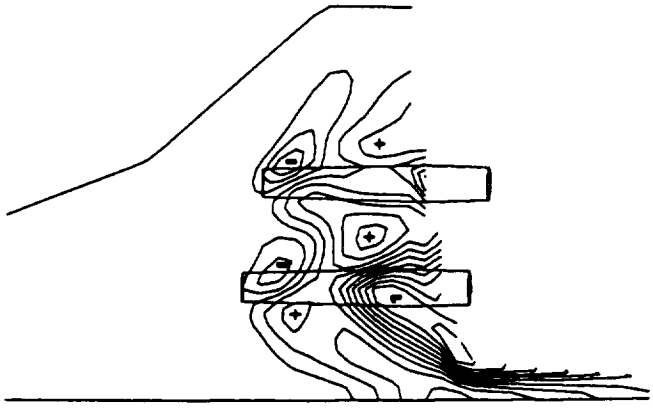

Figure 11. Pressure increments due to to flow-through nacelles. "+" or "-" indicates pressure above or below the wing-body case.

Performance

In order to support multidisciplinary optimization with practical turnaround times for design work, the code for analysis of each discipline must run quickly and scale efficiently on parallel machines. This section describes several aspects of performance, including single-processor computational rates, grid partitioning strategy, scalability, and choice of solution method for the pentadiagonal systems. All performance data reported are for 64-bit arithmetic and implementations based entirely on FORTRAN.

On a single i860 node, the sustained performance for ARC3D is about 6 MFLOPS, or 10% of the peak performance of the microprocessor. The primary cause of this degradation is the inadequate bandwidth and high latency for data movement between the chip's floating point registers and external memory. Another factor is the high cost of floating point divide operations and intrinsic functions such as square roots. The lack of efficient scheduling and pipelining of instructions by the still-evolving Fortran compilers also reduces computational rates. All megaflops rates quoted are calculated by comparing computing time per iteration on the iPSC/860 to the time on a Cray Y-MP. Operations counts from the Cray Hardware Performance Monitor are used. The actual number of floating point operations on the parallel machine is somewhat higher.

The scalability of the CFD module has been measured over a wide range of processor counts and grid sizes. The most favorable way of measuring scalability is to assume that the problem size will scale up with the

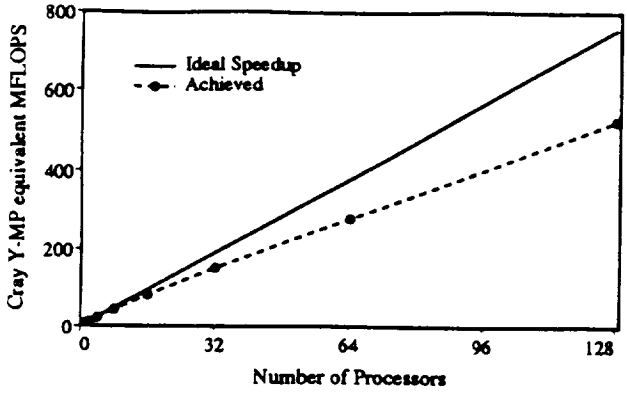

Figure 12. Performance of parallel ARC3D on the iPSC/860. Problem size is scaled with the number of processors.

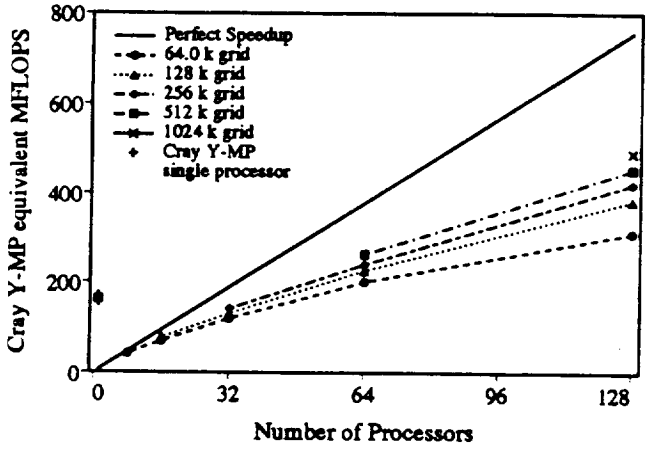

Figure 13. Performance of parallel ARC3D on the iPSC/860. Effect of spreading fixed-sized problems across additional processors. Cray Y-MP single-processor performance on the various grids is also shown.

number of processors available. The present code can compute up to about 14,000 grid points on each processor, given 8 Mb of memory per processor. Keeping the number of points near this maximum gives the results shown in Fig. 12. The "ideal speedup" curve indicates the speedup calculated by simply multiplying the computational rate of the CFD code on one processor by the number of processors. Let efficiency be the ratio of the actual processing rate to the "ideal speedup" case. By the time 128 processors are in use, efficiency has dropped to 70 percent, but the drop is gradual. At this point the code is operating at 527 megaflops. The cause of this performance degradation is the various types of parallel implementation overheads identified earlier. The cost associated with some of these overheads as a fraction of the total computational cost appears to grow at a superlinear rate as the total number of grid points and the number of processors increases.

In practice, grid sizes do not scale up indefinitely. Fig. 13 shows how performance varies when the number

Prob. Size	iPSC/860				Cray-YMP
		No. of Proc.			Time/step
	Algorithm	32	64	128	(MFLOPS)
(24x24x24)	CE-GE	0.23 (120)	0.14 (196)	0.11 (251)	0.22 sec. (123)
	PGE	0.29 (94)	0.19 (142)	0.13 (201)	
	SGE-BCR	0.47 (58)	0.32 (84)	0.22 (126)	
(40x40x40)	CE-GE	1.00 (136)	0.52 (260)	0.34 (401)	0.875 sec. (155)
	PGE	1.13 (120)	0.67 (202)	0.44 (310)	
	SGE-BCR	1.63 (83)	0.99 (136)	0.64 (211)	
(80x80x80)	CE-GE	-	3.99 (283)	2.05 (553)	6.85 sec. (165)
	PGE	-	4.31 (263)	2.50 (453)	
	SGE-BCR	-	5.71 (198)	3.38 (334)	
(160x80x80)	CE-GE	-	-	4.05 (558)	13.3 sec. (170)
	PGE	-	-	4.61 (490)	
	SGE-BCR	-	-	5.87 (385)	

Table 1. ARC3D performance with various algorithms for solution of pentadiagonal systems.

of processors is increased and grid size is held constant. In most cases, the efficiency drops by at least 10 percent for each factor-of-two increase in the number of processors. Thus, while performance scales well as grid size increases, the return for using more processors diminishes dramatically once the largest useful grid size is reached, and each processor has less computational work to do.

Table 1 shows the dependence of the time per step on the algorithms used to solve the multiple systems of pentadiagonal equations. On the iPSC/860, the best performance for any grid size and number of processors is obtained for the complete-exchange based implementations (CE-GE), while the sub-structured Gaussian elimination based algorithms (SGE-BCR) exhibit poor performance. This is primarily due to high arithmetic overhead associated with this class of algorithms, despite their high degree of easily exploitable concurrency. The pipelined Gaussian elimination based implementations (PGE) perform well, but appear to suffer from the relatively high message latency of the iPSC/860. Memory usage for these algorithms is calculated in 64-bit words per grid point: 67.5 for CE-GE, 44.5 for one-way PGE, and 49.5 for either two-way PGE or SGE-BCR. The calculation ignores the storage of overlap regions,

and counts integer arrays as one-half word. The data for results in Figs. 12 and 13 were obtained using the two-way PGE algorithm for solving pentadiagonal systems. The applications examples were computed with the one-way PGE scheme. The choice of algorithms was made largely on the basis of memory usage. The PGE methods allow the use of larger computational grids or fewer processors, compared to the faster CE-GE approach.

Processor Partition	Time/step	
	Problem Size	
$(P_i x P_j x P_k)$	(80x40x40)	(80x80x80)
(8x4x4)	0.76	2.72
(8x8x2)	0.88	3.01
(16x8x1)	1.06	3.96
(32x4x1)	1.30	6.47
(64x2x1)	2.10	-

Table 2. Effect of subdomain aspect ratio on performance.

The results in Table 2 show the dependence of time-per-step for a fixed grid size on the aspect ratio of the processor grid for a two-way pipelined Gaussian elimination based implementation. Optimum performance is obtained when the processor grid associated with the spatial data decomposition is proportional to the computational grid dimensions. Performance variations up to a factor of three can result from inappropriate spatial data decompositions. Similar results hold for implementation based on other pentadiagonal solution algorithms as well.

For practical use, the grid should be only large enough to resolve the flow physics with the required accuracy. For aircraft design, a range of 0.1 to 10.0 million grid points is reasonable. The real goal is not to reach some level of gigaflops or teraflops, but to reduce the time to obtain a solution. There will always be a point of diminishing returns in the use of large numbers of processors. Fig. 13 suggests that for practical grid sizes, processor counts in the hundreds, rather than thousands, will be most effective. Interprocessor communications and per-processor computing rates both play a part in determining the optimal number of processors. Both areas must improve substantially if low solution turnaround times are to be achieved. Faster communication can improve efficiency, allowing effective use of more processors, but the available speedup with 128 processors will be no more than a factor of 2 for most of the cases tested. Processors with higher sustained computational rates are essential, but in turn give a speedup which is limited by the communication latency, bandwidth, and network connectivity.

Another approach to high performance scalability is to do more than one problem at a time. Many design optimizers run at least one test case for each variable to be optimized, in order to calculate derivatives before stepping to a new design point. These cases can be run in parallel, with a considerable improvement in efficiency. For example, an optimization case might use a grid of 256,000 points. On 128 processors, the case would run at 420 MFLOPS (see Fig. 13). For four cases run concurrently on 32 processors each, the processing rate would be 142 x 4, or 568, MFLOPS, finishing 26 percent sooner.

Future Work

The CFD module will be combined with an optimizer, for use in aerodynamic optimization. Integration with structures and propulsion codes will proceed, to give the ability to analyze and optimize high speed aircraft at cruise conditions. Other disciplines, such as controls, will be included later, to broaden the range of flight conditions for which these tools are useful.

The parallelization of the code depends only on having a MIMD (Multiple Instruction, Multiple Data) computer with a message passing capability. Most current and emerging parallel computer architectures meet this description, and the code will be implemented on those which become available as HPCCP testbed machines. Each new implementation will be evaluated to ensure that the capabilities of these machines are used efficiently.

Conclusions

A new CFD module provides significant progress toward the goal of performing multidisciplinary computations on highly parallel computers. The module computes both Euler and Reynolds-averaged Navier-Stokes solutions about complex aircraft configurations. It covers flow speeds from takeoff, through the HSCT flight regime, to higher Mach numbers, provided perfect gas and continuity assumptions apply. An algebraic turbulence model and a wide selection of boundary conditions are included.

It is now possible to compute compressible flowfields with familiar tools, but on computer architectures which will scale to unprecedented levels of performance. This capability is available for both single-discipline fluids research, and for inclusion in multidisciplinary analysis and optimization.

There is substantial room for improvement in all areas affecting CFD performance on parallel computers. With nearly two orders of magnitude between the usable single-processor performance of the iPSC/860 and the best vector supercomputers, there is room for dramatic improvements. The cost-effectiveness of those improvements, particularly in the areas of memory access speed and interprocessor bandwidth, will be critical. User codes will improve gradually, as programming for parallel machines becomes better understood,

or perhaps more rapidly, if improved algorithmic approaches are discovered. Computation of single CFD problems at teraflops rates does not seem to be within reach, but teraflops multidisciplinary optimization may be only a few years away.

Acknowledgements

Computational resources were provided by the Numerical Aerodynamic Simulation (NAS) Program at NASA Ames Research Center. This research has been funded through NASA Ames Research Center Cooperative Agreement NCC 2-505 and Contract NAS 2-12961.

References

[1]Renze, K. J., Buning, P. G., and Rajagopalan, R. G., "A Comparative Study of Turbulence Models for Overset Grids," AIAA Paper 92-0437, January, 1992.

[2]Pulliam, T. H., and Chaussee, D. S., "A Diagonal Form of an Implicit Approximate-Factorization Algorithm," *Journal of Computational Physics*, Vol. 39, pp. 347-363, 1981.

[3]Holst, T. L., Thomas, S. D., Kaynak, U., Gundy, K. L., Flores, J., and Chaderjian, N. M., "Computational Aspects of Zonal Algorithms for Solving the Compressible Navier-Stokes Equations in Three Dimensions," *Numerical Methods in Fluid Mechanics I*, edited by K. Oshima, Inst. of Space and Astronautical Sciences, Tokyo, 1985, pp. 113-122.

[4]Ryan, J. S., Flores, J., and Chow, C.-Y., "Development and Validation of a Navier-Stokes Code for Hypersonic External Flow," *Journal of Spacecraft and Rockets*, Vol. 27, No. 2, 1990, pp. 160-166.

[5]Benek, J. A., Dougherty, F. C., and Buning, P. G., "Chimera: A Grid-Embedding Technique," AEDC-TR-85-64, December 1985.

[6]Suhs, N. E., and Tramel, R. W., "PEGSUS 4.0 User's Manual," AEDC-TR-91-8, June 1991.

[7]Meakin, R. L., "A New Method for Establishing Intergrid Communication among Systems of Overset Grids," AIAA-91-1586-CP, AIAA 10th Computational Fluid Dynamics Conference, June 14-27, 1991, Honolulu, Hawaii.

[8]Baldwin, B. S. and Lomax, H., "Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, January 1978.

[9]Barszcz, E., "Intercube Communication for the iPSC/860," Proceedings of the Scalable High Performance Computing Conference, pp. 307-313, Williamsburg, Virginia, April 1992.

[10]Ying, S. X., Steger, J. L., Schiff, L. B., and Baganoff, D., "Numerical Simulation of Unsteady, Viscous, High-Angle-of-Attack Flows using a Partially Flux Split Algorithm," AIAA Paper 86-2179, August 1986.

[11]Lawrence, S. L., Chaussee, D. S., and Tannehill, J. C., "Application of an Upwind Algorithm to the Three-Dimensional Parabolized Navier-Stokes Equations," AIAA paper 87-1112, June 1987.